UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

Ingeniería Superior de Telecomunicaciones

Proyecto Fin de Carrera

Etiquetado emocional de música en tiempo real

Autor: Antonio José Blanco Calle

Tutor: Ángel Navia Vázquez

Septiembre 2011

Agradecimientos

Aprovecho estas líneas para agradecer a Ángel Navia, tutor de este Proyecto Fin de Carrera, el apoyo, la atención y la comprensión que ha mostrado durante este último paso en la universidad.

Me gustaría dedicar este Proyecto Fin de Carrera a mi familia, mis padres Emilio y Toni, mis hermanos, Julio y Emilio, a mi cuñada Teresa, por su amor y apoyo incondicional durante todos estos años, por no dejarme caer, por estar siempre ahí. A las últimas en llegar, mis sobrinas Carmen y Teresa que, sin ser conscientes, han llenado nuestra familia de alegría y gracia. A mis queridos abuelos, Juan y Paco, a mi abuela Carmen –allá donde esté. A mi abuela Teresa, que de algún modo se enterará y se sentirá orgullosa. A mis tíos y primos. A todos, gracias por confiar en mí.

También se lo dedico a mis mejores amigos: Kiko y José Luis, uno no sabe qué caminos le depara el futuro, pero sabe que con amigos como ellos todo resultará más llevadero. A mis "amigos de Madrid", gente extraordinaria, a los que admiro, con los que me río y aprendo: Borja –sufridor colchero como un servidor, Carlos, Christian, Maxi, Elsa y Blanca. Gente fetén.

A Carmen e Irene, por estar siempre a mi lado. A mis "amigos del pueblo": Javi, las dos María José, Rocío y Mercedes, en los que siempre encuentro apoyo y comprensión. A los que fueron mis compañeros de piso: Fernando, Juan y Chema. Finalmente, a "mis amigos de Erasmus", que me hicieron pasar el mejor año de mi vida: Mika, Helen, Daniele, Linda y Sandra.

A todos ellos, muchas gracias.

Resumen

El objetivo final de este proyecto es realizar una aplicación bajo la plataforma iOS, presente en los dispositivos portátiles de Apple, que consiga la identificación instantánea del tipo de música que está sonando, asociando un estado emocional de entre unos predeterminados, en este caso: “alegre”, “triste” y “normal”. La aplicación que se pretende desarrollar incluirá todas las funcionalidades necesarias para tal efecto: etiquetado de los diferentes fragmentos de la canción, entrenamiento y clasificación. De este modo, el usuario no necesita recurrir a otros dispositivos más que a su propio iPod, iPhone o iPad. Adicionalmente, se ofrece en la aplicación la posibilidad al usuario de crear su propio clasificador, estableciendo las categorías que considere oportuno.

En esta memoria se contará paso a paso cómo se logró conseguir el objetivo propuesto. La estructura del documento concuerda con la estructura con que se llevó a cabo el propio proyecto. Primero, introduciremos los aspectos teóricos sobre las distintas tecnologías que se han utilizado: la extracción de coeficientes cepstrales de frecuencia Mel (MFCC) y su tratamiento, la clasificación por máquinas de soporte vectorial y el desarrollo de aplicaciones para iOS. Tras ello, explicamos cómo se llevo a cabo la implementación de la aplicación, desde la búsqueda de un modelo estadístico apropiado en Matlab hasta la programación bajo el entorno Xcode. Una vez concluida la parte de desarrollo de la aplicación, expondremos los resultados obtenidos y explicaremos las conclusiones extraídas de todo el proceso. Finalizaremos con la propuesta de nuevos trabajos que se sustenten en el explicado en este estudio.

Índice general

Índice de figuras

Índice de tabla

1. Introducción

Durante los últimos años, ha habido un interés creciente en el estudio de la extracción de características de la música. Desde la creación de formatos de audio de alta compresión, tales como el MP3, la industria de la música ha sufrido un cambio radical que la ha obligado a modificar su modelo de negocio. En cuestión de años se ha pasado del viejo modelo de ventas de CDs a la apertura de tiendas de música en la red, tales como la tienda de Amazon o la iTunes Store de Apple. Son muchas las ventajas que ofrecen los nuevos formatos a los usuarios: facilidad de transferencia, gran calidad, posibilidad de llevar toda tu biblioteca musical en el bolsillo, compra *online*…

Este nuevo escenario ha propiciado que millones y millones de canciones estén presentes en la red. Millones y millones de canciones que suponen una cantidad información inmensa. Y un reto. Tal cantidad de información puede a la vez ser una amenaza o una oportunidad, dependiendo de cómo sea tratada. Gracias al avance de las tecnologías de procesado de la información, es posible extraer características relevantes de archivos musicales. Así, es posible por ejemplo: obtener el género, medir la similitud con otras canciones, detectar estribillos, etc.

Por otro lado, el avance de los dispositivos portátiles ha permitido la creación de aparatos que, aparte de almacenar y reproducir archivos multimedia, cuentan con gran capacidad de procesamiento y facilidad de uso, permitiendo el desarrollo de aplicaciones para todo tipo de disciplinas.

1.1 Motivación

La motivación de este proyecto tiene doble vertiente: una basada en los intereses del mercado y otra poniendo énfasis en la comodidad del usuario.

El nuevo modelo de negocio musical se basa, en su mayoría, en la venta de música en formato digital. El usuario accede al portal del proveedor y descarga directamente los ficheros

de audio comprimido a su ordenador o dispositivo móvil. De esta manera, el proveedor es conocedor de los gustos de su cliente: sabe sus artistas, sus géneros favoritos, etc. Esto abre una oportunidad de mercado excelente, puesto que puede recomendar productos similares a tenor de las preferencias del cliente. Por ello, cualquier método para extraer características de archivos musicales es bienvenido.

Por otro lado, resulta interesante para el propio usuario cualquier método para clasificar su biblioteca. En la actualidad, no es extraño encontrarse bibliotecas musicales de varios miles de canciones – que suponen varias decenas de gigabytes de información. Por consiguiente, parece de gran ayuda encontrar un método para, no sólo clasificar bajo los parámetros clásicos (artista, género o incluso tempo, intensidad) sino permitir al usuario crear sus propias categorías.

Parece muy interesante encontrar un método sencillo y cómodo para el usuario de etiquetar y clasificar sus archivos musicales. Este es, como detallamos a continuación, el objetivo del proyecto que nos ocupa.

1.2 Objetivo del proyecto

El objetivo de este Proyecto Fin de Carrera es, principalmente, implementar una aplicación para los dispositivos iOS de Apple Inc. que permita al usuario clasificar su música según las características elegidas por él mismo. Sin embargo, el proyecto tiene varias particularidades:

- La clasificación se realiza con respecto a los fragmentos de las canciones, esto es, no se clasifica una canción entera sino que se hace con sus distintas partes. De este modo, cuando el usuario reproduce cierto fichero puede ver a qué categoría pertenece el fragmento que escucha en ese momento.
- En la aplicación se ha incluido un modelo de clasificación por defecto con tres categorías (“Alegre”, “Triste” y “Normal”) para que el usuario pueda comenzar a probar la aplicación nada más ejecutarla. Sin embargo, se brinda la posibilidad de crear un clasificador personalizado, para lo cual se ha creado una interfaz de etiquetado y entrenamiento dentro de la misma aplicación.
- La clasificación se sustenta en la asignación de las categorías a los coeficientes cepstrales de Mel (MFCC) obtenidos de cada fragmento.

El aspecto más interesante de este proyecto es poder crear un clasificador personalizado dentro de una aplicación de un dispositivo móvil. Todo el proceso para tal fin se ha

implementado dentro del programa: la selección de los ajustes del clasificador, el etiquetado, el entrenamiento y la clasificación.

1.3 Contenido de la memoria

El contenido de la memoria se divide en tres grandes bloques:

- En el primero se introducen los distintos conceptos sobre los que se sustentará la aplicación: la extracción de características de canciones, la integración temporal de dichas características, la clasificación y el desarrollo de aplicaciones para dispositivos portátiles.
- En el segundo se estudia la primera fase del desarrollo de la aplicación, bajo el entorno Matlab, donde se buscará el mejor modelo posible para ser plasmado finalmente en iOS.
- En este último bloque se tratarán todos los detalles de la implementación de la versión final de la aplicación que es objetivo de este proyecto.

Concluiremos la memoria con las conclusiones extraídas a lo largo de la realización del proyecto y con la propuesta de posibles extensiones a la aplicación desarrollada.

2. Estado de la cuestión

El sistema de clasificación de nuestra aplicación estará formado por tres bloques principales: en primer lugar situamos un extractor de características musicales –en nuestro caso coeficientes de Mel-, seguidamente nos encontramos con un integrador que permite reducir la cantidad de información que llega al último bloque, el clasificador. En la Figura 2.1 vemos una representación del sistema.

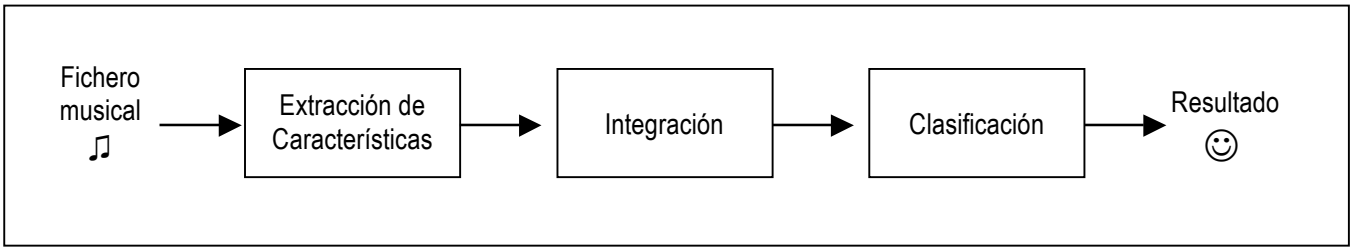

Figura 2.1. Esquema del sistema de clasificación.

Las características que extraeremos de los archivos musicales serán los Coeficientes Cepstrales de frecuencia Mel (MFCC), ampliamente reconocidos como herramienta para sintetizar información musical. Sin embargo, también introduciremos otros métodos alternativos y los compararemos. En segundo lugar, presentaremos los métodos utilizados para la integración en el tiempo de los vectores de características. La integración se hace necesaria por la gran cantidad de información generada en la etapa anterior de extracción. En nuestro caso, con el objetivo de crear una aplicación para una plataforma móvil, es de vital importancia reducir al máximo la carga computacional. En tercer lugar, trataremos los métodos de clasificación existentes, haciendo especial hincapié en la clasificación multiclase, y en la implementación que LIBSVM hace de ella. Por último, nos centraremos en el que es nuestro objetivo final: la implementación de una aplicación en el sistema operativo móvil de Apple, iOS. Para ello presentaremos su entorno de desarrollo.

2.1 Extracción de características de canciones

A continuación, se expondrán algunos de los métodos existentes para la extracción de características de audio. El estudio será breve para todos ellos excepto para el método de Coeficientes Cepstrales de frecuencia Mel (MFCC) puesto que es el más aceptado y el utilizado para el problema que nos ocupa.

2.1.1 Coeficientes Cepstrales de Frecuencia Mel (MFCC)

Los coeficientes cepstrales de frecuencia Mel (MFCC) intentan capturar la información perceptual del sonido. El algoritmo para su consecución fue originalmente propuesto por Davis y Mermelstein (1980) en un sistema de reconocimiento automático de voz. Actualmente tienen un gran número de aplicaciones, más allá del reconocimiento de voz, tales como la obtención de características musicales (clasificación de género, timbre…) o la medida de similitud de audio.

Los MFCC se basan en la escala de Mel, que tiene en cuenta el hecho de que la correlación entre la diferencia de tonos entre dos sonidos percibidos por el oído humano no es lineal en frecuencia. De hecho es más fácil para una persona distinguir dos sonidos de baja frecuencia , como por ejemplo 300 Hz y 400 Hz, que otros de alta frecuencia, como 6000 Hz y 6100 Hz, aún teniendo en ambos casos una separación en frecuencia de 100 Hz. Esto es debido a que el ancho de las bandas críticas tiene un crecimiento aproximadamente lineal hasta la frecuencia de 1 KHz y aproximadamente logarítmico a partir de ella. En la Figura 2.2 se muestra la relación entre la frecuencia en mels y en Hz [1].

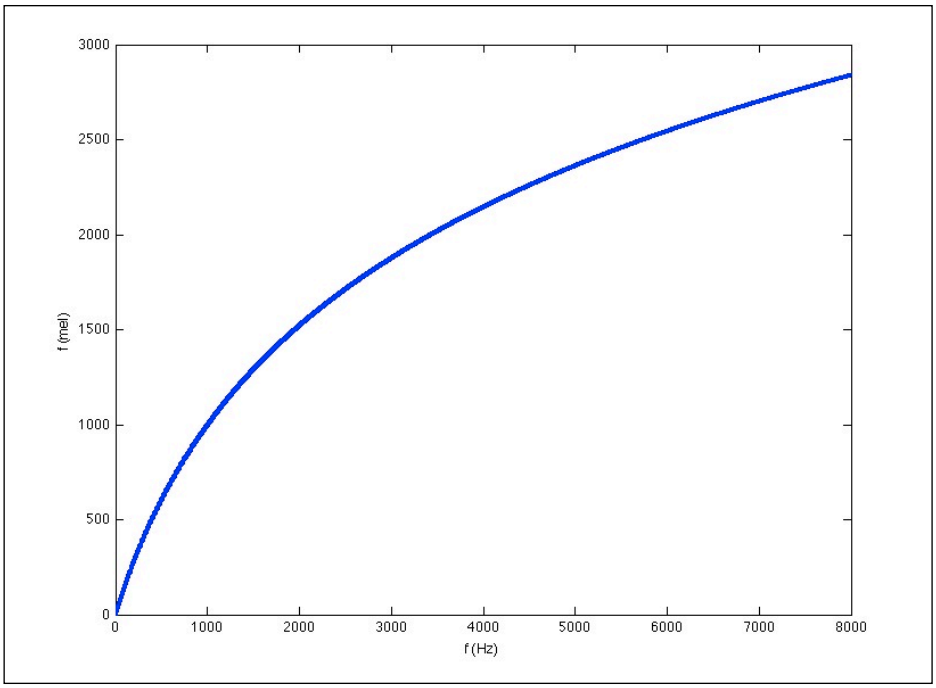

Figura 2.2. Frecuencias en Mels frente a frecuencias en Hz.

2.1.1.1 Procedimiento de extracción de los MFCC.

Para obtener los MFCC de un archivo de sonido debemos seguir los siguientes pasos [2]:

- Enventanado temporal de la señal.
- Cálculo de la transformada discreta de Fourier (DFT).
- Estimación de la Densidad Espectral de Potencia (DEP).
- Obtención del vector de coeficientes Mel (VCM).
- Obtención de coeficientes cepstrales MFCC.

A continuación, se explicará detalladamente cada paso del proceso enunciado anteriormente y representado en el diagrama de bloques de la Figura 2.3.

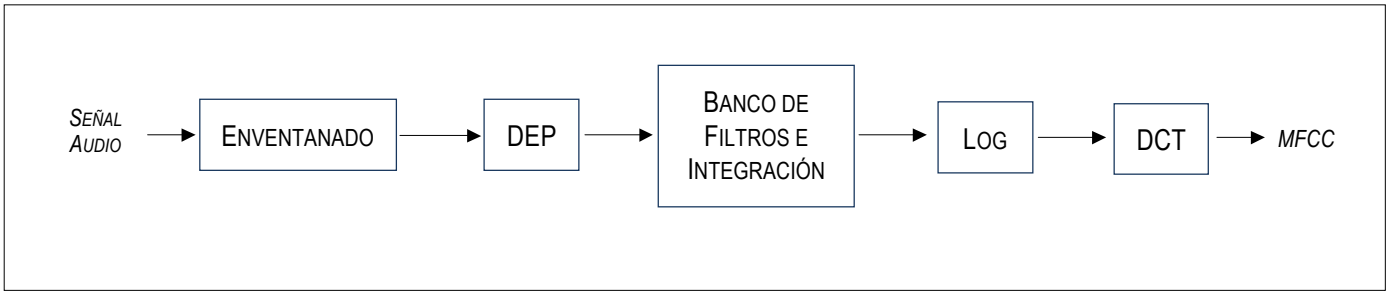

Figura 2.3. Diagrama de bloques del proceso de extracción de los MFCC.

Enventanado de la señal.

Mediante el enventanado conseguimos limitar la longitud de la secuencia que posteriormente vamos a transformar. El principal objetivo de este enventanado es obtener para cada fragmento unas características espectrales lo suficientemente estacionarias. Frecuentemente, el enventanado suele realizarse de manera solapada para así mantener las características de las transiciones entre ventanas.

La naturaleza de la ventana, su forma y su duración, determinará aspectos importantes para el proceso en general. Así, cuanto más rápidamente cambien las características de la señal, más corta (en tiempo) debería ser la ventana. Sin embargo, una longitud excesivamente corta podría limitar la resolución en frecuencia. Por lo tanto, es de vital importancia encontrar el tamaño de ventana adecuado para cada aplicación.

Existen multitud de formas de ventana, en nuestro caso utilizaremos una ventana tipo Hanning, que suaviza la señal en los extremos, eliminando discontinuidades generadas por aplicar la DFT a una señal no periódica –supresión del rizado o *ripple*.

Banco de filtros de Mel.

Habiendo conseguido una secuencia de duración finita a través del enventanado, se procede al cálculo de la transformada discreta de Fourier (DFT), cuya expresión corresponde a:

$$X(k) = \sum_{n=0}^{N-1} x(n) \exp(-j \frac{2\pi kn}{N}) \quad \textbf{[2.1]}$$

siendo *x(n)* la señal de audio tratada. El siguiente paso es calcular la Densidad Espectral de Potencia (DEP), a partir de *X(k)*:

$$DEP(k) = |X(k)|^2 \qquad \textbf{[2.2]}$$

Para caracterizar la señal de audio utilizaremos un banco de filtros triangulares solapados (Figura 2.4), cuyas bases están comprendidas entre las frecuencias centrales de sus filtros adyacentes. La separación de frecuencias centrales de cada filtro se elige de forma lineal de acuerdo a la escala Mel.

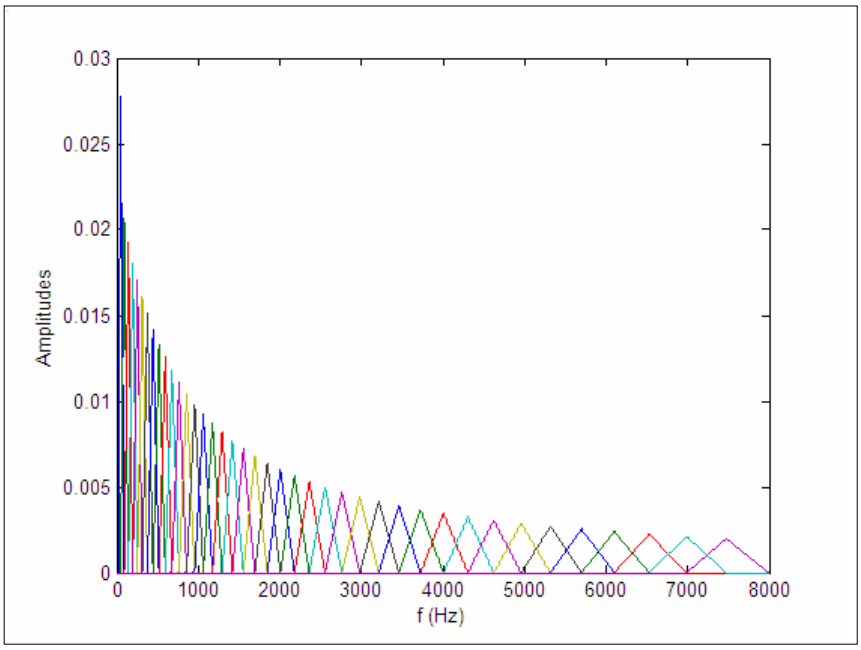

Figura 2.4. Banco de filtros de Mel.

La DEP de la señal es filtrada con el banco de filtros e integrada en cada una de las bandas, consiguiendo así el Vector de Coeficientes Mel, cuyos componentes representan la información de la DEP presente en cada banda, ponderada por la forma del filtro.

Obtención de los MFCC.

Antes de seguir, es necesario definir una nueva magnitud llamada *cepstrum*. El *cepstrum* se define como la Transformada Inversa de Fourier (IFT) del logaritmo de la Densidad Espectral de Potencia, esto es:

$$C(\tau) = IFT(\log(DEP(k))) \quad \textbf{[2.3]}$$

siendo τ una variable en el dominio *cepstral*, cuya unidad es la 'cuefrencia', anagrama de 'frecuencia', del mismo modo que 'cepstrum' y 'spectrum' (espectro, en castellano).

Puesto que la DEP es una función real y par:

$$C(\tau) = FT(\log(DEP(k))) \quad \textbf{[2.4]}$$

Los pretendidos coeficientes cepstrales de frecuencia Mel son una modificación de la anterior ecuación. Por un lado, el logaritmo se aplica sobre el Vector de Coeficientes Mel (VCM) y, por otro, se utiliza la Transformada Discreta del Coseno (DCT) en lugar de la Transformada de Fourier. Aplicando estas transformaciones a la ecuación anterior, obtenemos los coeficientes cepstrales de frecuencia Mel:

$$C(\tau) = DCT(10 * \log(VCM)) \quad [2.5]$$

Parámetros utilizados.

Tal y como hemos visto en esta sección, el algoritmo de extracción de MFCC es abierto en cuanto a la elección de sus parámetros. La correcta elección de los mismos nos brindará la posibilidad de obtener una configuración que proporcione los resultados deseados.

A continuación, examinamos los parámetros más influyentes a la hora de calcular los coeficientes cepstrales:

- **Tamaño de ventana**: El tamaño de ventana utilizado al analizar señales de audio depende de la aplicación que se va a realizar. En reconocimiento de palabras, por ejemplo, se utiliza un tamaño tal que permita el reconocimiento de los fonemas que componen la palabra, de unos 10 ms aproximadamente. Para aplicaciones relacionadas con búsqueda de similitudes en la música, como la que nos ocupa, se utilizan tamaños de ventana más amplios, en un rango de los 200 a los 2000 ms, con un solapamiento del 50 %.
- **Número de MFCC**: El número de coeficientes con que se caracteriza el espectro de una ventana determina con qué precisión se aproxima la Densidad Espectral de Potencia. En la mayoría de trabajos consultados, se utiliza un rango de 6 a 12 coeficientes de MFCC.
- **Número de filtros del banco y rango de frecuencias**: La resolución espectral depende del número de filtros del banco y del rango de frecuencias que cubre.

En nuestro caso, estos parámetros y su influencia sobre los resultados finales se estudian más adelante, en el capítulo 3.

2.1.2 Otros métodos de extracción de características.

El uso de los coeficientes cepstrales de frecuencia Mel para extracción de características de señales de audio se convirtió en estándar de facto desde la publicación del documento [3] en 1997. Sin embargo, no es el único método utilizado. A modo informativo, se incluyen en este estudio otros métodos que, si bien no fueron implementados para su prueba en el proyecto, consideramos que son de interés [4].

2.1.1.2 Vectores de Croma (VC).

El método de Vectores de Croma se basa en observar la señal como una secuencia de notas y comparar líneas melódicas. El procedimiento para obtener dicho vector comienza por el

enventanado –con ventanas Hanning –de bloques de N muestras. De cada bloque se extrae un vector de características de 12 componentes, correspondientes a los 12 semitonos de cada octava. Dichos coeficientes corresponden a los valores resultantes del filtrado de la DEP mediante un banco de filtros cromáticos –cada filtro corresponde con un semitono de las octavas analizadas- y la posterior integración del resultado para cada croma. El esquema seguido es el mostrado en la Figura 2.5 y, de forma analítica, en la Ecuación [2.6]

$$\boldsymbol{v(c) = \sum_{h=OCTl}^{OCTH} \sum_{k=1}^{N} BPF_{c,h}(k) DEP(k)} \text{ [2.6]}$$

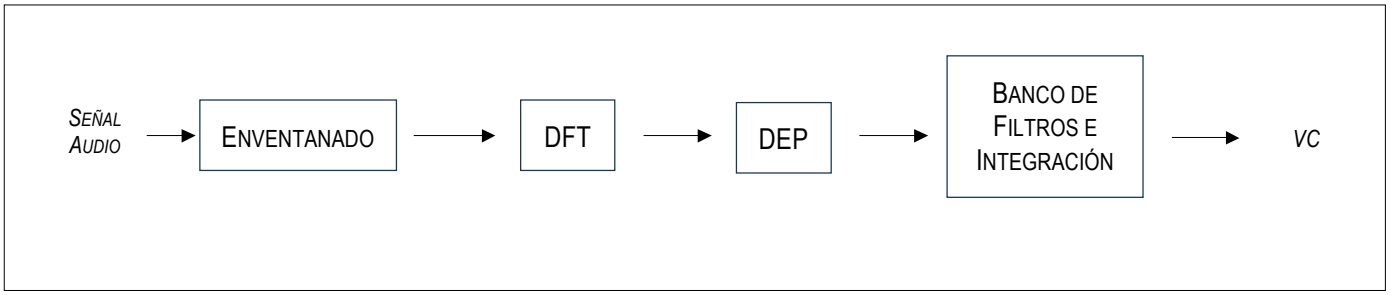

Figura 2.5. Diagrama del procedimiento de extracción de características mediante Vectores de Croma para cada bloque de muestras.

Este método compara parejas de fragmentos de aproximadamente un segundo de duración, pero a diferencia de MFCC, analiza el contenido melódico (semitonos) en lugar del timbre. Las bandas de frecuencia analizadas son las correspondientes a seis octavas de la escala cromática (130 Hz a 4 kHz). Se consideran los componentes de cada croma (12 coeficientes) sin distinguir entre las distintas octavas.

2.1.1.3 Transformada CQT

La extracción de información armónica directamente de espectrogramas conlleva muchos problemas, como los derivados por el hecho de la percepción logarítmica del oído humano frente a la resolución lineal de frecuencia de la Transformada de Fourier [5].

A diferencia de los métodos anteriores, la Transformada CQT no aplica directamente la DFT. Para obtener el vector de características, primero se realiza el enventanado de la señal. Tras ello, se aplica un banco de filtros sobre cada ventana para extraer los coeficientes de 36 semitonos distintos.

Con la ayuda de la ecuación situada más abajo ([2.7]), procedemos a explicar cómo calcular los coeficientes.

$$X(k) = \frac{1}{N_k} \sum_{n=0}^{N_k - 1} x(n) e^{\frac{-j2\pi Qn}{N_k}} \text{ [2.7]}$$

donde *X(k)* representa la energía espectral de la k-ésima nota, centrada en la frecuencia f_k para una ventana dada, donde a su vez:

$$f_k = f_0 \cdot 2^{k/12}, k = 0,1,2, \dots 35 \quad \textbf{[2.8]}$$

Se escoge f_0 = 130.8 Hz, que representa la nota DO en la tercera octava, bajo la hipótesis de que la música analizada tendrá frecuencias superiores a ésta. Q es una constante que relaciona la resolución temporal con la frecuencia y N_k es el ancho de ventana.

Este método analiza las bandas de la escala cromática distinguiendo los semitonos entre tres octavas; de esta forma, el vector de características está compuesto por 36 elementos cubriéndose las frecuencias de 130 Hz a 1kHz. Se representan los semitonos de una forma más exacta que en VC ya que se utilizan distintas resoluciones temporales para analizar distintas frecuencias, por los que tiene una resolución más fina en frecuencia.

2.2 Integración de características temporales para la clasificación de música

Un sistema de clasificación de música consta en su forma más básica de un bloque de extracción de características, seguido de un clasificador. Sin embargo, en ocasiones el número de vectores de características entregado por el primer bloque –en nuestro caso, vectores de coeficientes de Mel- es tan elevado que se hace necesaria la inserción de un nuevo bloque, el de integración (por ejemplo, para un archivo MP3 de 4:10 minutos de duración y 6,6 MB de tamaño, se obtiene un fichero de vectores de 1,4 MB de tamaño). La integración de características temporales consiste en combinar todos los vectores de características pertenecientes a la misma ventana temporal en uno solo con el fin de capturar toda la información relevante de la ventana. El tamaño de la ventana, generalmente fijo, determina la escala temporal de la característica tratada.

Si partimos de la observación de una secuencia temporal de características, x_i de dimensión D, la idea de la integración temporal se puede expresar analíticamente de la siguiente forma:

$$\boldsymbol{z_k = f(x_{k \cdot H_s}, \dots, x_{k \cdot H_s + W_s - 1})} \quad \textbf{[2.9]}$$

donde H_s y W_s configuran respectivamente el tamaño de salto y el de la propia ventana (ambos en número de muestras) , *k=0,1,..,* es el índice de la secuencia y *f(·)* es el método de integración utilizado. El tamaño de salto se define como el número de muestras que se desplaza la ventana temporal de integración. También es posible definirla en tanto por ciento con respecto al propio tamaño de la ventana.

La media y la varianza se antojan como el método más sencillo para comenzar el estudio, sin embargo, es inmediato reconocer que, mediante esta técnica no se tienen en cuenta ni los cambios en el tiempo de los datos ni las dependencias entre las distintas dimensiones del vector. Para solucionar este hándicap, estudiaremos los modelos de integración autorregresivos multivariable (en inglés, *multivariate autoregressive*) que, si bien conllevan una mayor carga computacional –como se verá en 2.2.4-, permiten modelar tanto las correlaciones temporales como la dependencia entre características.

Procedemos pues al estudio de cada uno de los métodos utilizados para integrar los vectores de MFCC. En el capítulo 3, se detallarán los resultados que arrojaron las pruebas realizadas a los diferentes métodos hasta decidir qué sistema es el idóneo para ser implementado en el sistema objeto de este proyecto.

2.2.1 Media-Varianza (Mean-Var)

Es un método sencillo y muy usado ([6],[7]) para la integración temporal de características, sobretodo en el campo de la clasificación musical por géneros. Este modelo se sostiene sobre tres asunciones: que las muestras son independientes entre ellas, que se rigen bajo distribución gaussiana y que las dimensiones de cada muestra son independientes entre ellas.

Los parámetros de este modelo son:

$$\begin{cases} \boldsymbol{m}_k = \frac{1}{W_s}\sum_{n=0}^{W_s-1} \boldsymbol{x}_{k\cdot H_s+n} \\ c_{i,k} = \frac{1}{W_s}\sum_{n=0}^{W_s-1}\left(x_{i,k\cdot H_s+n} - m_{i,k}\right)^2 \end{cases} \quad \textbf{[2.10]}$$

con *D* como la dimensión del vector de características, *i = 1,..., D.* y donde los vectores $\boldsymbol{m_k}$ y $\boldsymbol{c_k}$ representan los vectores estimación de la media y la varianza de los datos, respectivamente. Así

pues, el nuevo vector de características, tras la integración, es aquél de dimensión *2D* que se muestra en [2.11]:

$$\boldsymbol{z}_k = \begin{bmatrix} \boldsymbol{m}_k \\ \boldsymbol{c}_k \end{bmatrix} \quad \textbf{[2.11]}$$

Como podemos ver, este método presenta la nada desdeñable ventaja de ser fácil de implementar. Sin embargo, como se demuestra en [8], las premisas de independencia entre características y entre muestras son falsas. Como respuesta al problema, se plantea el estudio del siguiente método, el de la Media-Covarianza.

2.2.2 Media-Covarianza (Mean-Cov)

Este método tiene en cuenta la correlación entre las dimensiones de cada vector de características. De ahí que sea necesario pasar de un modelo gaussiano unidimensional a otro multidimensional, basado en una distribución de la forma $x \sim G(\boldsymbol{m}, \boldsymbol{C})$ donde el vector de medias y la matriz de covarianzas se calculan sobre la ventana temporal. El nuevo vector resultado de la integración por este método tendrá la siguiente forma [2.12]:

$$\boldsymbol{z}_k = \begin{bmatrix} \boldsymbol{m}_k \\ vech(\boldsymbol{C}_k) \end{bmatrix} \quad \textbf{[2.12]}$$

Donde $\boldsymbol{m_k}$ es el vector de medias –como el hallado para el modelo Media-Varianza- y *vech*($\boldsymbol{C}_k$) es un vector formado apilando la triangular superior –incluyendo la diagonal- de la matriz de covarianzas. La dimensión resultante pues del vector es *(D/2)(3 + D)*. Nótese que, al incluir la diagonal de la matriz de covarianzas, incluimos el vector de varianzas tal y como hacíamos en el método anterior.

El método de la Media-Covarianza deja sin resolver el problema de la relación temporal entre muestras, que será solucionado gracias a los métodos descritos a continuación.

2.2.3 Modelos Autorregresivos Multivariable: MAR y DAR

Los modelos autorregresivos multivariable vienen a solucionar los problemas planteados por los dos métodos anteriores: mientras que el método de Media-Varianza no modela ningún tipo de correlación entre muestras ni entre dimensiones, el método de Media-Covarianza sólo tiene en cuenta la correlación entre dimensiones, dejando fuera del modelo la relación entre muestras temporales. En esta sección se estudiará cómo los métodos basados en modelos

autorregresivos multivariable (en inglés *MAR*, *multivariate autoregressive model*) nos proporcionan un medio de integración que modele las correlaciones temporales de los datos.

Comenzaremos el estudio por el método más general, el MAR, para después particularizar y llegar hasta el DAR (del inglés, *diagonal autoregressive*) que, si bien no resuelve del todo el problema –asume independencia entre las dimensiones del vector- es útil como método de integración alternativo a los ya presentados. Es importante reseñar que el estudio que se realizará sobre los citados métodos no es exhaustivo, puesto que no son objeto del proyecto que nos ocupa. Para obtener un conocimiento más detallado, es altamente recomendable leer el trabajo de [8].

Para incluir tanto la correlación entre dimensiones como entre muestras, utilizamos el siguiente modelo analítico

$$\boldsymbol{x}_n = \sum_{p=1}^{P} \boldsymbol{A}_p \boldsymbol{x}_{n-I(p)} + \boldsymbol{u}_n \quad \textbf{[2.13]}$$

donde $\boldsymbol{u_n}$ representa el término de ruido, supuesto independiente e idénticamente distribuido de media v y matriz de covarianza $\boldsymbol{C}$. Por su parte, $I(p)$ define el conjunto de vectores pasados sobre los que se realizará la predicción actual. Generalmente, se utiliza $I= \{1,2, ...,P\}$. Las matrices $\boldsymbol{A_p}$ para $p = 1, ..., P$ son las matrices de orden P del modelo autorregresivo y determinan cuánta información de las salidas anteriores ($\{ \boldsymbol{x_{n-I(1)}}, \boldsymbol{x_{n-I(2)}}, ..., \boldsymbol{x_{n-I(P)}}\}$) está presente en la salida actual $\boldsymbol{x_n}$.

Existen varios métodos para estimar los parámetros del modelo autorregresivo, tanto en el dominio del tiempo como en el de la frecuencia. El más usado –y el que usaremos –es el método de mínimos cuadrados, que minimiza el error cuadrático medio. De este modo, estimaremos las matrices $\{\boldsymbol{A_1}, ...,\boldsymbol{A_P}\}$, la media v y la matriz de covarianza del ruido $\boldsymbol{C}$. Finalmente, conseguimos el vector integrado de la ventana temporal k para el modelo más general, el MAR, que resulta

$$\boldsymbol{z}_k = \begin{bmatrix} vec(\boldsymbol{B}_k) \\ \boldsymbol{v}_k \\ vech(\boldsymbol{C}_k) \end{bmatrix} \quad \textbf{[2.14]}$$

donde $B=[A_1, A_2, ..., A_P]$ es de dimensión D x PD y z_k, $(P + ½) D^2 + (3/2)D$. Para el modelo **DAR**, sólo utilizaremos las diagonales de $\boldsymbol{A_p}$ y $\boldsymbol{C}$.

2.2.4 Complejidad de los modelos

En la sección 3.4.2 de este proyecto se presenta un estudio de las prestaciones entregadas por cada uno de los métodos, así como del tiempo de ejecución de los mismos. De momento, introducimos la complejidad computacional de los métodos tal y como nos muestra [8].

La siguiente tabla (Tabla 2.1) muestra el número de multiplicaciones y sumas de cada uno de los métodos para una ventana temporal, donde D es la dimensión del vector de coeficientes, P es el orden del modelo DAR/MAR, y W_s es el tamaño de la ventana en número de muestras. En este estudio no se tuvo en cuenta el efecto del solapamiento entre ventanas temporales.

Método	**Multiplicaciones y sumas**
Media-Varianza	$4DW_s$
Media-Covarianza	$(D + 3)\ DW_s$
DAR	$(D/3)(P + 1)^3 + ((P + 6)(P + 1) + 3)\ DW_s$
MAR	$1/3\ (PD + 1)^3 + ((P + 4 + 2/D)(PD + 1) + (D + 2))\ DW_s$

Tabla 2.1.Complejidad computacional de los métodos de integración.

2.3 Clasificación con Máquinas de Soporte Vectorial

El último elemento en nuestro sistema de clasificación musical es el propio clasificador. En esta sección explicaremos las distintas alternativas con las que nos encontramos a la hora de implementar esta pieza fundamental. En concreto, hablaremos de las redes neuronales artificiales y de las máquinas de soporte vectorial. Tras compararlas, explicaremos más detenidamente la herramienta –LIBSVM –que ha hecho decantarnos por las máquinas de soporte vectorial. Finalmente, explicaremos los métodos de clasificación multiclase existentes, enunciando el utilizado en este proyecto.

Un problema de clasificación normalmente conlleva separar los datos en conjuntos de entrenamiento y de test. Cada elemento del conjunto de entrenamiento lleva asociada una etiqueta que determina la clase a la que pertenece y unos atributos, que son los datos en sí. Existen multitud de algoritmos y aproximaciones para llevar a cabo la clasificación. Las más utilizadas en la actualidad son las redes neuronales artificiales o ANN (del inglés, *Artificial Neural Network*) y las máquinas de soporte vectorial o SVM (del inglés, *Support Vector Machine*).

2.3.1 Redes Neuronales Artificiales

Las redes neuronales consisten en una simulación de las propiedades observadas en los sistemas neuronales biológicos a través de modelos matemáticos recreados mediante mecanismos artificiales (como un circuito integrado, un ordenador o un conjunto de válvulas). El objetivo es conseguir que las máquinas den respuestas similares a las que es capaz de dar el cerebro que se caracterizan por su generalización y su robustez [9].

Una red neuronal se compone de unidades llamadas neuronas. Cada neurona recibe una serie de entradas a través de interconexiones y emite una salida. Esta salida viene dada por tres funciones:

- Una **función de propagación**, que por lo general consiste en el sumatorio de cada entrada multiplicada por el peso de su interconexión (valor neto). Si el peso es positivo, la conexión se denomina excitatoria; si es negativo, se denomina inhibitoria.
- Una **función de activación**, que modifica a la anterior. Puede no existir, siendo en este caso la salida la misma función de propagación.

- Una **función de transferencia**, que se aplica al valor devuelto por la función de activación. Se utiliza para acotar la salida de la neurona y generalmente viene dada por la interpretación que queramos darle a dichas salidas. Algunas de las más utilizadas son la función sigmoidea (para obtener valores en el intervalo [0,1]) y la tangente hiperbólica (para obtener valores en el intervalo [-1,1]).

2.3.2 Máquinas de Soporte Vectorial

El objetivo de las máquinas de soporte vectorial es obtener un modelo (basado en el conjunto de entrenamiento) que prediga a qué clase pertenecen los elementos del conjunto de test a partir de los atributos de éstos. Las máquinas de soporte vectorial son un conjunto de algoritmos de aprendizaje supervisado desarrollados por Vladimir Vapnik y su equipo en los laboratorios de AT&T (a finales de los años 90). Conforman una red estática basada en *kernels* que realiza clasificación lineal sobre vectores transformados a un espacio de dimensión superior, esto es, separa mediante un hiperplano en el espacio transformado. Las operaciones que realiza una máquina de soporte vectorial se pueden resumir en tres:

- Transforma los datos a un espacio de dimensión muy alta a través de una función *kernel*. De este modo se reformula el problema de tal forma que los datos se mapean implícitamente en este espacio.
- Encuentra el hiperplano que maximiza el margen entre dos clases.
- Si los datos no son linealmente separables encuentra el hiperplano que maximiza el margen y minimiza el número de clasificaciones incorrectas.

De forma analítica, dado un conjunto de entrenamiento formado con los pares objeto-etiqueta $(\boldsymbol{x}_i, y_i)$, con $i = 1, \ldots, l$ donde $x_i \in R^n$ e $\boldsymbol{y} \in \{1, -1\}^l$, la máquina de soporte vectorial se basa en la solución del siguiente problema de optimización [10]:

$$\min_{w,b,\xi} \quad \frac{1}{2}\boldsymbol{w}^t\boldsymbol{w} + C\sum_{i=1}^{l}\xi_i \qquad \textbf{[2.15]}$$

$$sujeto\ a \quad y_i(\boldsymbol{w}^T\phi(\boldsymbol{x}_i) + b) \geq 1 - \xi_i, \qquad \xi_i \geq 0$$

Los vectores $\boldsymbol{x}_i$ son transformados a un espacio de mayor dimensión –puede que de dimensión infinita –por la función ϕ. La SVM encuentra un hiperplano con el máximo margen de separación. $C > 0$ es la penalización por error cometido. $K(\boldsymbol{x}_i, \boldsymbol{x}_j) \equiv \phi(\boldsymbol{x}_i)^T\phi(\boldsymbol{x}_j)$ es la llamada función *kernel*. Las funciones *kernel* más usadas son las siguientes:

- Lineal: $K(\boldsymbol{x}_i, \boldsymbol{x}_j) \equiv \boldsymbol{x}_i^T\boldsymbol{x}_j$.

- Polinómico: $K(\boldsymbol{x}_i, \boldsymbol{x}_j) \equiv \left(\gamma \boldsymbol{x}_i^T \boldsymbol{x}_j + r\right)^d,\ \gamma > 0.$
- RBF (del inglés, *radial basis function*): $K(\boldsymbol{x}_i, \boldsymbol{x}_j) \equiv \exp\left(-\gamma \left\|\boldsymbol{x}_i - \boldsymbol{x}_j\right\|^2\right),\ \gamma > 0.$
- Sigmoide: $K(\boldsymbol{x}_i, \boldsymbol{x}_j) \equiv \tanh\left(\gamma \boldsymbol{x}_i^T \boldsymbol{x}_j + r\right)$

Donde γ, r y d son parámetros del *kernel*.

2.3.3 Comparación

En la siguiente tabla, se resumen los principales aspectos comparativos de los dos métodos.

	Redes Artificiales Neuronales	**Máquinas de Soporte Vectorial**
Dimensión	Transformación a espacios de cualquier dimensión	Transformación a espacios de dimensión superior
Mínimos	Múltiples mínimos locales	Único mínimo global
Entrenamiento	Costoso	Muy eficiente
Clasificación	Muy eficiente	Muy eficiente
Parámetros de diseño	Número de capas, nodos	Kernel, coste

Tabla 2.2 Comparación de ANN vs SVM

A simple vista, ambos métodos son muy parecidos, pareciendo difícil la elección entre uno u otro. Sin embargo, existen varios factores que nos hacen decantarnos por las máquinas de soporte vectorial:

- La eficiencia en el entrenamiento hace más liviana la tarea de probar con distintos modelos, sobre todo si disponemos de una vasta cantidad de datos de entrenamiento.
- Las máquinas de soporte vectorial presentan gran robustez para generalizar.

- Por último, aunque quizás más importante, el hecho que nos decanta a utilizar las máquinas de soporte vectorial, es la existencia de una herramienta software que implementa todas las posibilidades de este método, se trata de la librería LIBSVM.

Por estos motivos, el modelo de clasificación que se utilizará será el de SVM. En el siguiente subapartado estudiamos la librería LIBSVM y las posibilidades que nos brinda para el desarrollo de este proyecto

2.3.4 LIBSVM

LIBSVM es una librería software para máquinas de soporte vectorial ampliamente adoptada. El propósito de la misma, creada por Chih-Chung Chang y Chih-Jen Lin, es proporcionar una herramienta que permita el uso de SVM en las aplicaciones y los estudios de los usuarios. Implementa clasificación de vectores soporte (C-SVC, un-SVC), regresión (épsilon-SVR, nu-SVR), estimación de distribuciones y clasificación multiclase. Además, ofrece un conjunto de métodos como la validación cruzada, distintos *kernels*, asignación de costes para datos sin balancear…

Aparte de todas las características mencionadas anteriormente, lo que hace a LIBSVM más atractiva, si cabe, es la posibilidad de poder integrarla bajo el software matemático MATLAB. De este modo, al ya de por si amplio abanico de posibilidades de MATLAB, podemos añadir la posibilidad de trabajar de manera fácil y completamente compatible con máquinas de soporte vectorial.

Veamos ahora qué elementos en particular de LIBSVM son convenientes para nuestro proyecto. Recordemos que la aplicación constará de dos opciones en cuanto a clasificación se refiere. Por un lado, se puede hacer uso del clasificador por defecto, que distingue entre tres categorías excluyentes, es decir, un clasificador multiclase. Por otro, se ofrece la opción de crear un clasificador a la medida del usuario, dando a elegir el número de categorías entre 1 (caso trivial) y 5. De todos los casos posibles, el que merece especial mención es el de dos categorías, pues contaríamos con varias formulaciones diferentes bajo el paraguas de LIBSVM. A continuación vamos a estudiar los diferentes casos.

2.3.4.1 Clasificadores de dos categorías

C-SVC (del inglés, C-Support Vector Classification)

Dado un conjunto de vectores de entrenamiento $\boldsymbol{x}_i \in R^n, i = 1, \dots, l$, que pertenecen a dos clases, y un vector $\boldsymbol{y} \in R^l$ tal que $y_i \in \{1, -1\}$, el método C-SVC resuelve el problema expuesto por la Ecuación de 2.15 transformándolo en

$$\min_\alpha \ \frac{1}{2}\boldsymbol{\alpha}^t Q\boldsymbol{\alpha} + e^T\boldsymbol{\alpha} \quad \textbf{[2.16]}$$

$$sujeto\ a \quad \boldsymbol{y}^T\boldsymbol{\alpha} = 0, \qquad 0 \le \alpha_i \le C$$

donde *i=1, ..., l*, e es el vector unidad, $C > 0$ es el límite superior, Q es una matriz *l*x*l* semidefinida positiva, $Q_{ij} \equiv y_i y_j K(\boldsymbol{x}_i, \boldsymbol{x}_j)$, y $K(\boldsymbol{x}_i, \boldsymbol{x}_j) \equiv \phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x}_j)$ es el *kernel* [11]. Los vectores $\boldsymbol{x}_i$ son mapeados a un espacio de dimensión superior por la función ϕ. Obteniendo la siguiente función de decisión

$$sgn\left(\sum_{i=1}^{l} y_i \alpha_i K(\boldsymbol{x}_i, \boldsymbol{x}) + b\right) \quad \textbf{[2.17]}$$

v-SVC (del inglés, v -Support Vector Classification)

Este método utiliza un nuevo parámetro *v* que controla el número de vectores de soporte y de errores de entrenamiento. El parámetro $\nu \in (0,1]$ es un margen superior para los errores de entrenamiento e inferior para el número de vectores de soporte [11].

Dado un conjunto de vectores de entrenamiento $\boldsymbol{x}_i \in R^n, i = 1, \dots, l$, que pertenecen a dos clases, y un vector $\boldsymbol{y} \in R^l$ tal que $y_i \in \{1, -1\}$, el método *v* -SVC resuelve el problema expuesto por la Ecuación de 2.15 transformándolo en

$$\min_\alpha \ \frac{1}{2}\boldsymbol{\alpha}^t Q\boldsymbol{\alpha} \quad \textbf{[2.18]}$$

$$sujeto\ a \quad \begin{cases} 0 \le \alpha_i \le C, i = 1, \dots, l, \\ \boldsymbol{e}^T\boldsymbol{\alpha} \ge \nu, \qquad \boldsymbol{y}^T\boldsymbol{\alpha} = 0 \end{cases}$$

donde $Q_{ij} \equiv y_i y_j K(\boldsymbol{x}_i, \boldsymbol{x}_j)$, obteniendo la siguiente función de decisión

$$sgn\left(\sum_{i=1}^{l} y_i \alpha_i K(\boldsymbol{x}_i, \boldsymbol{x}) + b\right) \quad \textbf{[2.19]}$$

Se ha demostrado (Crisp and Burges, 2000; Chang y Lin, 2001), que $\boldsymbol{e}^T\boldsymbol{\alpha} \ge \nu$ puede reemplazarse por $\boldsymbol{e}^T\boldsymbol{\alpha} = \nu$. Por lo que, en LIBSVM se resuelve una versión escalada de la Ecuación 2.18.

$$\min_\alpha \ \frac{1}{2}\boldsymbol{\alpha}^t Q\boldsymbol{\alpha} \quad \textbf{[2.20]}$$

$$sujeto\ a\ \begin{cases} 0 \leq \alpha_i \leq 1, i = 1, \dots, l, \\ \boldsymbol{e}^T\boldsymbol{\alpha} = \nu, \qquad \boldsymbol{y}^T\boldsymbol{\alpha} = 0 \end{cases}$$

Obteniendo a la salida α/ρ , por lo que la función de decisión es:

$$sgn\left(\sum_{i=1}^{l} y_i \left({}^{\alpha_i}/_{\rho}\right)(K(\boldsymbol{x}_i, \boldsymbol{x}) + b)\right) \quad \textbf{[2.21]}$$

y resultando como márgenes

$$y_i(\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_i) + b) = \pm 1 \quad \textbf{[2.22]}$$

que coinciden con los de C-SVC.

2.3.4.2 Clasificadores multiclase.

Como hemos avanzado, en la mayoría de ocasiones, el clasificador de nuestra aplicación va a tratar con más de dos clases. Por consiguiente, es apropiado detallar cómo funciona la librería LIBSVM en este escenario, así como introducir las diferentes aproximaciones.

El problema de la clasificación multiclase se refiere a la asignación a cada una de las observaciones a una de las *k* clases presentes. Generalmente, como los problemas de dos clases son mucho más sencillos, muchos autores proponen clasificadores de este tipo para el caso multiclase.

Existen dos aproximaciones para el problema de la clasificación multiclase: "uno contra todos" y "uno contra uno". Un clasificador "uno contra todos" es entrenado para separar una clase de las *K-1* restantes. El método estándar consiste en crear *K* clasificadores binarios, tantos como clases en paralelo. El tiempo de entrenamiento de este método es linealmente proporcional al número de clases. Todos los clasificadores son entrenados sobre el conjunto completo, lo que conlleva un mayor coste computacional, pero tiene la ventaja de contar con toda la información en la fase de entrenamiento de cada nodo. Por otro lado, tenemos la estrategia conocida como "uno contra uno" , que es usada por LIBSVM y que se basa en la construcción de *k(k – 1)/2* clasificadores, que son entrenados con dos categorías distintas. Para entrenar la clase *i* y la clase *j*, se resuelve el siguiente problema de clasificación de dos clases:

$$\min_{w^{ij}, b^{ij}, \xi^{ij}} \ \frac{1}{2}\left(\boldsymbol{w}^{ij}\right)^t \boldsymbol{w}^{ij} + C\left(\sum_t \left(\xi^{ij}\right)_t\right) \quad \textbf{[2.23]}$$

$$sujeto\ a\ \begin{cases} \left(\boldsymbol{w}^{ij}\right)^T \phi(\boldsymbol{x}_t) + b^{ij} \geq 1 - \xi_t^{ij}, si\ \boldsymbol{x}_t\ pertenece\ a\ la\ clase\ i \\ \left(\boldsymbol{w}^{ij}\right)^T \phi(\boldsymbol{x}_t) + b^{ij} \leq 1 - \xi_t^{ij}, si\ \boldsymbol{x}_t\ pertenece\ a\ la\ clase\ j \\ \xi_i \geq 0 \end{cases}$$

Para clasificar, LIBSVM utiliza una estrategia de votación: cada clasificador binario se considera un votante de modo que, un punto $\boldsymbol{x}$ será considerado de la clase *L* que más votos

haya recibido –conocido como votación por mayoría simple. Existen alternativas a la votación por mayoría simple: por mayoría absoluta (más de la mitad de los votos) y por unanimidad (todos los votos). En caso de empate, mismo número de votos, se escoge la clase con índice de clase menor [12].

2.4 Desarrollo de aplicaciones en dispositivos móviles

2.4.1 Introducción

Desde los gigantes y pesados terminales instalados en coches a los dispositivos que caben en el bolsillo, los teléfonos móviles han recorrido un largo viaje. De costar miles de euros, pesar casi un kilogramo y tener una batería de 60 minutos de duración, han pasado a valer en torno a cien euros, pesar alrededor de cien gramos e incluir un ordenador, una cámara de video, reproducción de video y audio y conexión a Internet. Estos nuevos dispositivos, conocidos como *smartphones* (palabra inglesa compuesta *smart*, inteligente, y *phone*, teléfono) han supuesto no sólo una revolución tecnológica sino también social. Lo que hace especiales a estos dispositivos es su capacidad de concentrar tantas características un terminal tan pequeño y portátil: reproducción multimedia, cámara de fotos, pantalla multi-táctil de alta resolución, navegación GPS, Wi-Fi, potentes procesadores y unidades gráficas [13].

Otro de los aspectos clave para el éxito de los *smartphones* es su software. Si bien los antiguos teléfonos móviles contaban con un sistema embebido, los actuales funcionan bajo un sistema operativo, que proporciona más flexibilidad en cuanto a posibilidades de uso. En la siguiente tabla (Tabla 2.3) se presentan los sistemas operativos para móviles más comunes hoy en día, así como sus características más importantes [14].

Sistema Operativo	**Compañía**	**Arquitectura CPU**	**Lenguaje de Programación**	**Tienda de Aplicaciones**	**Plataforma de desarrollo**
iOS	Apple	ARM	C, C++, Objective-C	App Store	Mac OS X
Android	Google	ARM, MIPS, Power Architecture, x86	C, C++, Java	Android Market	Multiplatform
WebOS	HP, Palm	ARM	C (Alba)	App Catalog	Multiplatform
Windows Mobile	Microsoft	ARM	C++	Windows Phone Marketplace	Windows
Blackberry OS	RIM	ARM	Java	App World	Windows

Symbian	Symbian Foundation	ARM	C++	Ovi Store	Windows

Tabla 2.3 Comparativa de los Sistemas Operativos Móviles en el mercado.

Otro de los aspectos ventajosos de los sistemas operativos móviles es la posibilidad de instalar aplicaciones de terceros. Esto es una gran noticia para todas las partes: para las empresas –de telefonía o no, porque pueden proporcionar servicios al cliente de una forma más directa; para el usuario, que puede beneficiarse de un gran abanico de posibilidades y para el propio fabricante, pues puede conseguir beneficios al actuar de intermediario a través de las tiendas virtuales de aplicaciones. Además, como hemos visto en la Tabla 2.3, la mayoría de los S.O. (Sistemas Operativos) ofrecen a los desarrolladores, un conjunto de herramientas para la creación de aplicaciones, los llamados SDK (del inglés, *Software Development Kit*).

De entre todas las opciones que existían para el desarrollo de este proyecto, se escogió la plataforma iOS de Apple, por varios motivos:

- **Calidad del SDK**: Proporciona un conjunto de herramientas fáciles de usar (API, programas de escritura de código…), aparte de un simulador muy realista.
- **Número de desarrolladores**. Según los estudios, en 2010, el porcentaje de desarrolladores de aplicaciones para iOS significaba el 80% del total, seguido por el 20% de Android [15]. Esta supremacía desemboca en una mayor cantidad de fuentes de ayuda y códigos de ejemplo para el sistema operativo móvil de Apple, lo cual hace mucho más llevadera la tarea de desarrollar aplicaciones propias.
- **Cuota de mercado**. Si bien, los dispositivos con iOS no dominan la cuota de mercado, el crecimiento en apenas cuatro años ha sido destacable (ver Tabla 2.4 y Figura 2.6) [16]. Además si tenemos en cuenta que la tienda de aplicaciones de Apple (App Store) está mostrándose más beneficiosa en cuanto a ingresos respecto a su principal competidora Android Market (alrededor de 20 USD gastados por usuario y mes en la App Store frente a los 14 USD de Android Market, según un estudio de Mobclix, [17]), podemos concluir que el de Apple es el sistema operativo móvil más apropiado para llevar a cabo nuestro proyecto.

Año	Symbian	Android	Blackberry	iOS	Microsoft	Otros
2007	63,5%	N/A	9,6%	2,7%	12,0%	12,1%
2008	52,4%	0,5%	16,6%	8,2%	11,8%	10,5%
2009	46,9%	3,9%	19,9%	14,4%	8,7%	6,1%
2010	37,6%	22,7%	16,0%	15,7%	4,2%	3,8%
2011–Q1	27,4%	36,0%	12,9%	16,8%	3,6%	3,3%

Tabla 2.4 Cuota de mercado de los diferentes SO móviles desde el año 2007 hasta el primer cuarto de 2011

A continuación hablaremos con más detenimiento de la plataforma de desarrollo proporcionada para la creación de aplicaciones para iOS.

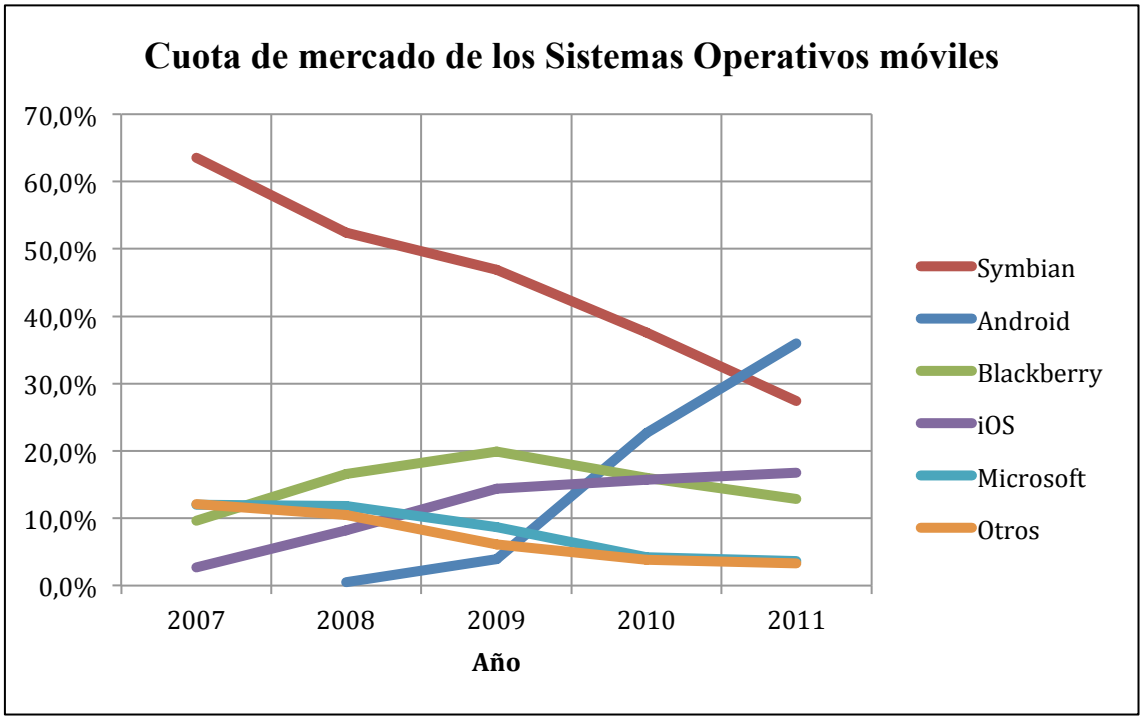

Figura 2.6 Cuota de mercado de los Sistemas Operativos móviles

2.4.2 Desarrollo de aplicaciones para iOS

iOS es el sistema operativo que se ejecuta en los terminales *iPhone*, *iPod Touch* y *iPad*. Este sistema operativo se encarga de controlar el *hardware* y proporciona las tecnologías necesarias para desarrollar nuevas aplicaciones nativas, así como un conjunto de herramientas ya desarrolladas, como telefonía, correo electrónico y navegación web [18].

El SDK contiene las herramientas e interfaces necesarias para desarrollar, instalar, ejecutar y probar las mencionadas aplicaciones nativas, que son creadas bajo los *frameworks* (marcos, en castellano) incluidos en iOS y gracias al lenguaje de programación *Objective-C*.

2.4.2.1 Arquitectura de iOS y *frameworks*

iOS actúa de intermediario entre el hardware del dispositivo y las aplicaciones que aparecen en la pantalla. En lugar de interactuar con el hardware directamente, las aplicaciones utilizan un conjunto de interfaces del sistema que protegen a la aplicación de posibles cambios

de hardware. De este modo, es más fácil crear aplicaciones consistentes en dispositivos con diferentes características hardware.

La arquitectura de iOS se puede concebir como un conjunto de capas (mostradas en la Figura 2.7). En las capas inferiores se encuentran los principales servicios sobre los que se sustenta la aplicación, en las capas superiores se encuentran los servicios y las tecnologías más específicas.

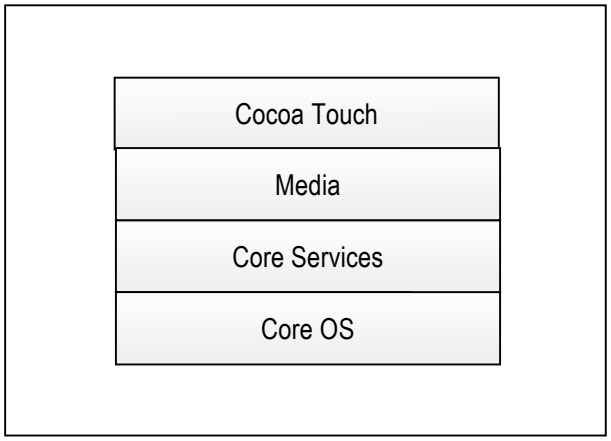

Figura 2.7 Capas de la arquitectura iOS

A la hora de escribir una aplicación es preferible utilizar *frameworks* de alto nivel a aquellos de bajo nivel. Un *framework* es un directorio que contiene una librería dinámica compartida (y sus recursos) y que puede ser utilizada para la creación de aplicaciones. Los *frameworks* de alto nivel proporcionan abstracciones de alto nivel para funciones de bajo nivel. Por lo general, estas abstracciones facilitan la escritura de código puesto que reducen la cantidad de código y encapsulan funciones potencialmente complejas. A continuación, introducimos un breve repaso de las diferentes capas del sistema, enumerando sus servicios y sus *frameworks*.

Cocoa Touch

Esta capa contiene los *frameworks* para la creación de aplicaciones para iOS. En ella se definen la infraestructura básica de la aplicación así como el soporte para tecnologías claves como la multi-tarea, el reconocimiento de gestos para la interfaz de usuario, la impresión, la protección de datos, las notificaciones, etc.

Para ello Cocoa Touch incluye los siguientes *frameworks* y servicios (ver Tabla 2.5).

COCOA TOUCH	
Framework	**Servicios**
Address Book UI Framework	Operaciones con la agenda de contactos: crear, editar…
Event Kit UI Framework	Operaciones con elementos relacionados con el calendario.
Game Kit Framework	Conectividad *peer-to-peer* para jugar en red y compartir partidas.
iAd Framework	Permite insertar anuncios en la aplicación.
Map Kit Framework	Proporciona una interfaz para navegar por mapas.
Message UI Framework	Permite la composición y recepción de correos electrónicos y SMS.
UIKit Framework	Proporciona la infraestructura clave para la implementar aplicaciones gráficas orientadas a eventos en iOS. Está presente en todas las aplicaciones.

Tabla 2.5 Frameworks de la capa Cocoa Touch

Media

Contiene las tecnologías relacionadas con la inclusión de gráficos, audio y video en la aplicación.

MEDIA	
Framework	**Servicios**
Assets Library Framework	Interfaz para la obtención de fotos y vídeos del terminal.
AV Foundation Framework	Captura y reproducción de vídeos y audio.
Core Audio Framework	Manipulación de audio.
Core Graphics Framework	Interfaz para la creación de gráficos
Core MIDI Framework	Proporciona una interfaz de comunicación con dispositivos MIDI.
Core Text Framework	Herramientas para el tratamiento de texto
Core Video Framework	Crea un buffer para la reproducción de vídeo, nunca se utiliza directamente.
Image I/O Framework	Interfaz para la importación/exportación de imágenes y sus metadatos.

Media Player Framework	Pone a disposición del desarrollador la interfaz nativa de reproducción de vídeo y de audio.
OpenAL Framework	Herramienta para tratar audio.
OpenGL ES Framework	Herramientas para dibujar gráficos en 2D y 3D
Quartz Core Framework	Contiene Core Animation, que es una tecnología de animación avanzada y composición.

Tabla 2.6 Frameworks de la capa Media

Core Services

Contiene los servicios de sistema fundamentales que todas las aplicaciones necesitan para su funcionamiento. Si bien, no suelen usarse de manera directa, muchos elementos del sistema se construyen sobre ellas. Algunas de las funciones más destacadas son: procesamiento multi-hilo, gestión de bases de datos, soporte para XML, etc.

Los *frameworks* perteneciente a esta capa y los servicios que ofrecen se muestran de forma resumida a continuación.

CORE SERVICES	
Framework	**Servicios**
Address Book Framework	Acceso programático a los contactos guardados en el dispositivo.
CFNetwork Framework	Conjunto de funciones para trabajar con protocolos de red.
Core Data Framework	Tecnología para gestionar una aplicación basada en Modelo-Vista-Controlador.
Core Foundation Framework	Interfaz que proporciona herramientas para la gestión de datos y servicios.
Core Location Framework	Herramientas relacionadas con la localización.
Core Media Framework	Recoge los formatos multimedia utilizados por AV Foundation Framework.
Core Telephony Framework	Interfaz para el uso de funciones de telefonía.
Event Kit Framework	Proporciona acceso a los eventos de los calendarios en el dispositivo.
Foundation Framework	Proporciona soporte a Core Foundation Framework

Mobile Core Services Framework	Define los tipos de bajo nivel usados en UTI (del inglés Uniform Type Identifiers).
Quick Look Framework	Interfaz para la previsualización de documentos
Store Kit Framework	Soporte para la compra de contenidos y servicios tanto en las tiendas virtuales de la compañía como en las aplicaciones de terceros.
System Configuration Framework	Conjunto de interfaces utilizadas para la configuración y ajuste del dispositivo.

Tabla 2.7 Frameworks de la capa Core Services

Core OS

La capa Core OS contiene las prestaciones de bajo nivel que el resto de tecnologías presentes utilizan. Normalmente se utilizan de forma implícita, salvo en el escenario de la comunicación con accesorios externos o con problemas relacionados con seguridad.

CORE OS	
Framework	**Servicios**
Accelerate Framework	Interfaces para cálculos matemáticos complejos y DSP
External Accessory Framework	Soporte para la comunicación con dispositivos externos, ya sea vía conexión con puerto *dock* o vía Bluetooth.
Security Framework	Herramientas para añadir seguridad extra a las aplicaciones.
System	Recoge el *kernel*, los *drivers* y las interfaces de bajo nivel UNIX.

Tabla 2.8 Frameworks de la capa Core OS

2.4.2.2 Herramientas del SDK

El SDK de iOS incluye todas las interfaces, herramientas y recursos necesarios para desarrollar aplicaciones para iOS bajo la plataforma Mac OS X. Además de los *frameworks* tratados anteriormente, se incluyen unas librerías compartidas. Esto se debe a que iOS está basado en UNIX, por lo que mucha de las funciones que forman la parte más baja del sistema operativo provienen de funciones de código abierto. Por lo que muchas de estas funciones están disponibles para ser usadas.

El entorno de programación incluido en el SDK de iOS está formado por tres herramientas:

- **Herramientas Xcode**:
  - **Xcode**: Entorno de desarrollo integrado que gestiona tu aplicación y que te permite editar, compilar, ejecutar y probar tu código. A su vez Xcode se integra con otras herramientas, como Interface Builder e Instruments.
  - **Interface Builder**: Herramienta que permite diseñar la interfaz de usuario de la aplicación. En la actualidad, está incluida dentro de la aplicación Xcode.
  - **Instruments**: Herramienta para el análisis de la ejecución y prueba de las aplicaciones. Es útil para medir el comportamiento en tiempo real de tu aplicación e identificar posibles problemas.
- **iOS Simulator:** Aplicación de Mac OS X que simula el funcionamiento de un dispositivo iOS, por lo que permite la prueba de tus aplicaciones de forma local en tu ordenador.
- **iOS Developer Library:** Documentación de referencia para los desarrolladores.

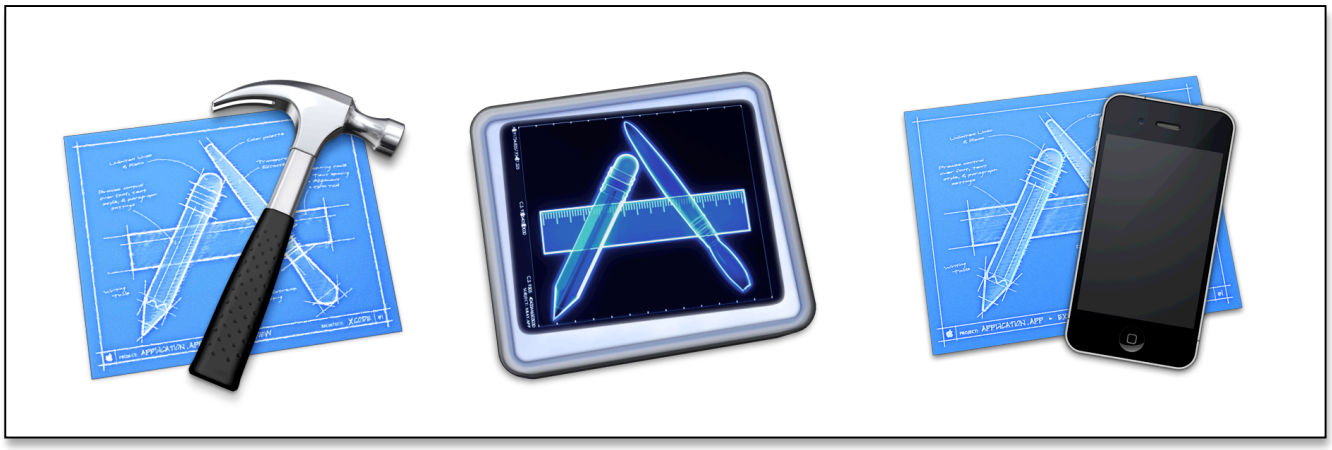

Figura 2.8 Iconos de las aplicaciones Xcode, Instruments y iOS Simulator, respectivamente.

2.4.2.3 Objective-C

El lenguaje de programación utilizado para desarrollar aplicaciones en iOS es Objective-C. Se trata de un lenguaje orientado a objetos que surgió como una extensión del estándar ANSI C. La mayoría de nuevas características se basan en Smalltalk, uno de los primeros lenguajes

orientados a objetos. Objective-C se diseñó para dar a C la capacidad de orientación a objetos de una manera sencilla [19] [20].

Como la mayoría de entornos orientados a objetos, consta de varias partes:

- Un lenguaje orientado a objetos.
- Una librería de objetos.
- Un conjunto de herramientas de desarrollo.
- Un entorno de ejecución.

Puesto que está basado en C, a la hora de implementar la aplicación, el compilador acepta código tanto de C como de C++. Este hecho será de vital importancia para el desarrollo de este proyecto, como se verá más adelante

3. Desarrollo del modelo en Matlab

Una vez introducidas las diferentes tecnologías y conceptos que servirán de cimiento para la creación de la aplicación que es objetivo de este proyecto, procedemos a detallar las distintas fases del desarrollo. Desde el primer momento, se creyó oportuno utilizar el software Matlab para dar los primeros pasos, en lugar de encarar directamente el problema en el entorno de desarrollo de iOS. Matlab ofrece flexibilidad a la hora de programar funciones, aparte de multitud de ellas ya programadas. Además, afronta cálculos complicados de manera más solvente que lenguajes no diseñados para tal efecto.

Los capítulos 3 y 4 narran el proceso seguido para desarrollar la aplicación final. El capítulo 3 –en el que nos encontramos –comienza describiendo los datos con los que trabajaremos: los ficheros musicales. Seguidamente, trataremos la parte del desarrollo seguido en Matlab, en la que intentamos hacer una primera aproximación al modelo más adecuado para implementar. Para ello, fue necesario empezar de cero: crear un etiquetador, entrenar los datos y seleccionar los parámetros más convenientes. Una vez conseguido el modelo, hablaremos en el capítulo 4 sobre todo lo concerniente a la aplicación iOS: estructura, interfaz de usuario, modelo utilizado, etc.

3.1 Base de datos

Antes de adentrarnos en los trabajos realizados en Matlab es necesario detallar qué tipo de archivos de música utilizaremos como datos de entrenamiento y de test. En total, 83 canciones se han usado para el proyecto, 60 de ellas forman el conjunto de entrenamiento y las 23 restantes, el de test. Se consideró suficiente este número de canciones, también se intentó

incluir en los conjuntos canciones de géneros diferentes (pop, rock, clásica…) para dar mayor robustez a la aplicación. En el Apéndice B se listan las canciones utilizadas.

En cuanto a la naturaleza de los ficheros, enumeramos sus principales características a continuación:

- Formato: 'Waveform Audio Format' ('.wav').
- Duración: 60 segundos.
- Canales: Mono.
- Bits por muestra: 16.
- Tasa de muestreo: 16.000 Hz

Todos los archivos utilizados se obtuvieron a través de la conversión de archivos '.mp3' a '.wav' con las características mencionadas arriba mediante la aplicación iTunes de Apple. Si bien el formato '.mp3' es el más extendido hoy en día para la codificación de archivos musicales, se creyó conveniente el uso de ficheros '.wav' por la rapidez con que Matlab procesa los ficheros '.wav' –a través de la función 'wavread' –mucho mayor que para los ficheros '.mp3', que deben ser decodificados –de hecho, la función 'mp3read' hace uso de la función 'wavread'. La ausencia de compresión de los archivos '.wav' nos proporciona la ventaja del procesado rápido pero nos castiga con un tamaño de archivo excesivo (para una canción en formato '.mp3' de duración 3:30 y tamaño 4,4 MB ,obtenemos un archivo '.wav' de 38,5 MB a 48 KHz, estéreo y 1536 kbps). Para contrarrestar este efecto, decidimos eliminar un canal –de estéreo a mono-, y reducir la tasa de muestreo a 16.000 Hz, ya que de este modo reducimos el tamaño del fichero sin perder la calidad necesaria para la aplicación (con la canción del ejemplo anterior, obtenemos un tamaño de archivo de 8,9 MB, con lo que reducimos unas cuatro veces el tamaño del primer '.wav' y solamente duplicamos el tamaño del '.mp3' original).

3.2 Desarrollo de la interfaz para el etiquetado en Matlab

La necesidad de crear una interfaz para el etiquetado de canciones surge de la inexistencia en el mercado de un software con esta funcionalidad. Las aplicaciones existentes relacionadas crean o modifican las etiquetas ID3, estándar de facto para incluir metadatos, pero en ningún caso te permite incluir dentro del fichero información de las características según el fragmento de la canción en particular.

Así, si lo que se desea es crear una extensa base de datos sobre características de determinados segmentos de canciones es recomendable tener el apoyo de una herramienta que facilite un trabajo que, de otra manera, se antoja tedioso.

Aunque el lenguaje más habitual para crear interfaces es Java, se optó por la creación en Matlab, que no sólo ofrece más herramientas para las operaciones matemáticas sino que además entrega un mayor rendimiento a la hora de hacer cálculos. Dentro de Matlab, se hizo uso de la herramienta GUIDE (en español, Entorno de Desarrollo de Interfaces Gráficas de Usuario) que conforma un juego de herramientas para la creación de interfaces gráficas de usuario (en inglés, GUI) de manera intuitiva y rápida. Aunque es posible crear la interfaz directamente desde el editor de funciones de Matlab, la herramienta GUIDE proporciona un entorno de desarrollo muy visual que propicia una fácil implementación, evitando el farragoso trabajo de crear elementos gráficos desde llamadas a funciones.

3.2.1 Estructura de la Interfaz

La interfaz se compone de una única ventana a través de la cual podemos configurar las opciones y ejecutar la aplicación. Se pretendió que la interfaz fuera lo más intuitiva posible, por ello se decidió incluir simplemente lo necesario para un correcto etiquetado. En la Figura 3.1 se muestra una captura de la ventana principal de la interfaz, cuyos componentes procedemos a explicar.

Figura 3.1. Ventana principal de la interfaz 'gui.gui'

3.2.1.1 Panel Canción

Es el primer panel de la ventana (empezando desde arriba). En él se muestran los detalles más relevantes de la canción que se va a etiquetar: Tasa de muestreo, Bits por muestra y Ruta, que muestra la localización del archivo que va a ser tratado.

Dentro del panel, en la esquina superior derecha, se encuentra el botón "Cargar Canción", que es utilizado para elegir, a través de una ventana de diálogo del sistema, el archivo '.wav' a etiquetar.

3.2.1.2 Panel Categorías

Es el segundo panel de la ventana, situado bajo el panel "Canción". En él establecemos las categorías que se van a tener en cuenta a la hora de etiquetar.

El primer elemento del panel es un cuadro de texto editable, precedido de la etiqueta "Introduzca el número de categorías (de 1 a 4)" , en el que introduciremos el número de categorías de nuestro clasificador.

Más abajo, nos encontramos con unas filas formadas por un conjunto texto editable y botón. En el texto editable, con etiqueta "Nombre de categoría 1 (opcional)", el usuario introducirá opcionalmente el nombre dado a la categoría. El botón situado a la derecha servirá para etiquetar un fragmento de la canción según la categoría a la que acompaña. El nombre de la categoría es opcional y meramente orientativo puesto que no se verá reflejado en los resultados finales. Su inclusión responde al deseo de facilitar el uso al usuario.

Por último, en la esquina inferior derecha del panel se encuentra el indicador de "Última categoría etiquetada", cuya función es recordar al usuario cuál fue la última categoría que éste asignó a un segmento de la canción.

3.2.1.3 Botón Comenzar

Por último, en la parte inferior de la ventana se encuentra el botón comenzar que, tras ser pulsado, activa la reproducción de la canción y el mecanismo de etiquetado.

3.2.2 Funcionamiento de la Interfaz

El funcionamiento de la interfaz es sencillo e intuitivo, simplemente hay que rellenar las opciones desde arriba de la ventana principal hasta su final. El diagrama de bloques de la Figura

3.2 muestra el flujo de operación de la aplicación. En esta sección explicaremos de forma detallada los pasos que hay que seguir para un etiquetado exitoso.

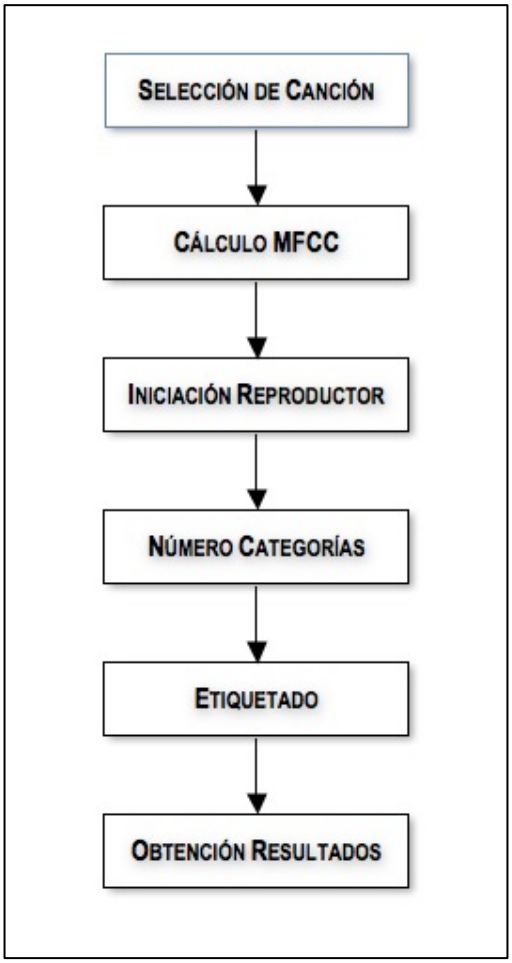

Figura 3.2 Diagrama de bloques del proceso de etiquetado

3.2.2.1 Selección de la canción

El primer paso es seleccionar la canción que deseamos etiquetar. Para ello, es necesario pulsar el botón "Cargar Canción", que nos mostrará una ventana de selección de fichero del Sistema Operativo (Figura 3.3). Como comentamos anteriormente, el formato de los ficheros que vamos a utilizar es '.wav', es por ello que la ventana de diálogo está limitada a la selección de ficheros de este tipo, sombreando el nombre del resto de ficheros.

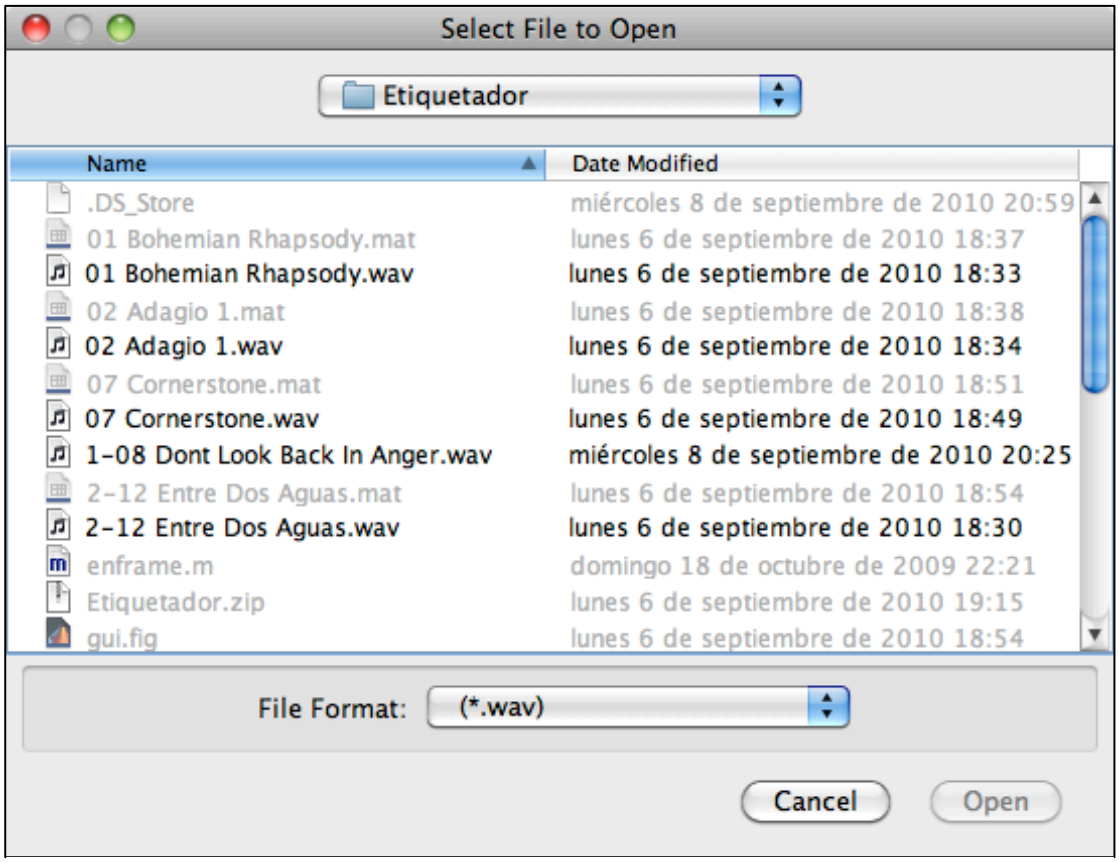

Figura 3.3. Ventana de selección de archivo

3.2.2.2 Cálculo de los MFCC y creación del reproductor de sonidos

Una vez cargada la canción (seleccionada y habiendo pulsado el botón "Open"), se calculan los coeficientes MFCC, mediante una llamada a una función externa. El cálculo de los coeficientes es independiente de la interfaz, esto es, podrían utilizarse distintas implementaciones de la extracción de coeficientes sin que esto afectase al núcleo de funcionamiento de la interfaz. Al cambiar de implementación, sólo hay que tener en cuenta cuántos vectores por segundo de canción se obtienen, parámetro que está determinado por el tamaño de ventana.

En este punto, se aprovecha la llamada a la función de lectura del archivo '.wav' (función 'wavread'), para la creación del reproductor de audio que utilizaremos después, a través de la función 'audioplayer', que recibe como parámetros de entrada el vector con la onda de sonido y la frecuencia de muestreo.

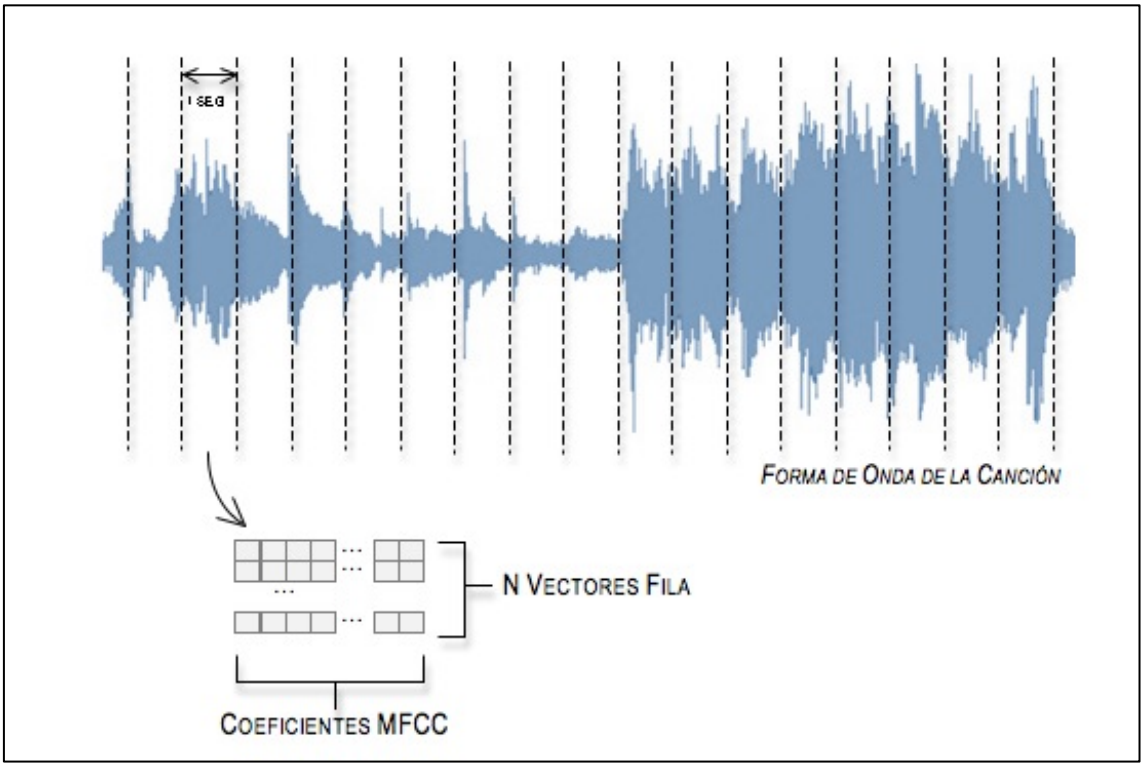

Figura 3.4. Esquema de extracción de coeficientes por fragmento de canción

Para el cálculo de los MFCC, se ha utilizado una versión simplificada del programa diseñado a tal efecto en la herramienta conocida como Toolbox. La función que calcula los coeficientes tiene como parámetros de entrada la propia señal de audio, la frecuencia de muestreo y el número de coeficientes deseado. Como salida, entrega un una matriz donde cada fila es un vector de coeficientes de Mel. La dimensión de la matriz depende del número de coeficientes elegido (número de columnas) y del enventanado. En el caso que nos ocupa utilizaremos una ventana Hanning de 16 ms de duración –que equivale a ventanas de 256 muestras (16.000 muestras/s x 16 ms), generando aproximadamente 125 vectores de coeficientes por segundo, siguiendo el siguiente procedimiento:

$$Nº\ de\ ventanas = \left\lfloor \frac{L - l_v + t_s}{t_s} \right\rfloor \ donde \begin{cases} L \equiv longitud\ de\ la\ señal\ de\ audio\ [muestras] \\ lv \equiv longitud\ de\ la\ ventana\ [muestras] \\ t_s \equiv longitud\ del\ salto\ [muestras] \end{cases} \quad [3.1]$$

para L = 960.000 muestras, l_v = 256 y t_s = 128 (50 %), es inmediato que

$$Nº\ de\ ventanas = 7499$$

Como todas nuestras canciones tienen 60 s de duración obtenemos que la tasa de vectores por segundo es

$$Nº\ de\ vectores\ por\ segundo = \left\lfloor \frac{Nº\ de\ ventanas}{duración\ canción} \right\rfloor = 125 \frac{vectores}{s} \quad [\,3.2\,]$$

Este valor nos será de gran utilidad a la hora de realizar el etiquetado pues nos ayuda a discernir qué vectores pertenecen a qué categoría según el momento en que el usuario pulsa el botón de categoría correspondiente. El tamaño de la ventana debe ser en todo caso menor de 40 ms, porque sólo para estos tamaños se considerará que la asunción de estacionariedad es correcta. Se consideró que 16 ms era un valor apropiado. El único parámetro cuyo valor óptimo es objeto de estudio es el número de coeficientes, trataremos esto más adelante (apartado 3.4).

3.2.2.3 Introducción del número de categorías

Es necesario para la ejecución de la aplicación introducir el número de categorías que se desea etiquetar (Figura 3.5). El número de categorías por defecto es 0, si se intentara ejecutar con esta cifra, se mostraría por pantalla un mensaje de error alertándonos de la situación. El número máximo de categorías es cuatro, valor escogido de manera arbitraria y que podría modificarse sin mayor complicación.

Figura 3.5. Cuadro de texto para establecer el número de categorías

Cuando ya se ha establecido el número de categorías, es posible empezar el etiquetado, sin embargo, el usuario tiene la opción de dar nombres a dichas categorías, teniendo esta acción una intención más orientativa que funcional. Como se explicó en la sección anterior, los botones que acompañan al nombre de la categoría y etiquetados con números de 1 a 4 (Figura 3.6), tienen la función de establecer que un fragmento pertenece a una categoría u otra, una vez son pulsados.

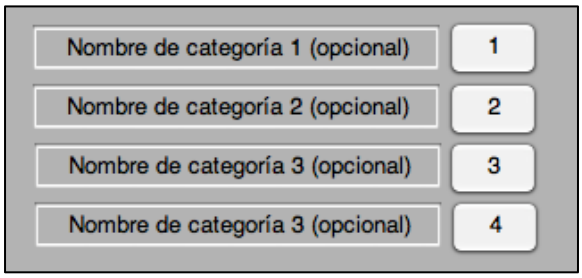

Figura 3.6. Cuadros de texto para introducir el nombre de cada categoría y botones para el etiquetado

3.2.2.4 Etiquetado

Una vez se han completado los pasos anteriores satisfactoriamente, se procede al etiquetado de la canción. Para ello, se ha de pulsar el botón "Comenzar", que inicia la reproducción de la canción –gracias a la función 'play' de Matlab sobre la estructura de nombre 'player' creada anteriormente.

Es en esta fase en la que el usuario debe decidir a qué categoría pertenece cada fragmento. El proceso de etiquetado sigue los siguientes pasos:

1) El usuario escucha cierto fragmento de una canción y decide a qué categoría pertenece (por ejemplo, Categoría 1).
2) Cuando cree que el fragmento asociado a dicha categoría (Categoría 1) ha terminado y da paso a una categoría distinta (digamos, Categoría 2), pulsa el botón asociado a Categoría 1.
3) Todos los vectores fila asociados a ese fragmento (vector siguiente al último etiquetado hasta vector correspondiente al momento en que se pulsa el botón) serán etiquetados según la Categoría 1.
4) Se vuelve a 1) hasta que la canción llega a su fin, momento en que el usuario decide a qué categoría pertenece el último fragmento sin etiquetar.
5) Cuando todos los vectores han sido etiquetados, se finaliza la ejecución, entregando el fichero con los resultados.

La Figura 3.7 muestra el proceso de etiquetado de una forma gráfica, con el fin de explicar de una manera más clara los diferentes pasos.

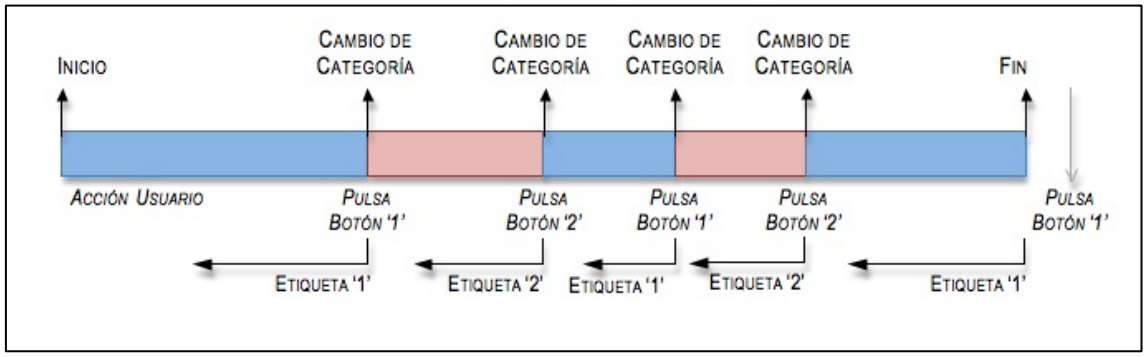

Figura 3.7. Ejemplo de etiquetado

3.2.3 Formato de los resultados

El resultado del etiquetado es un archivo '.mat' del mismo nombre que el fichero que contiene la canción, compuesto por los vectores fila que contienen los coeficientes MFCC, acompañados por una columna que almacena la categoría que ha sido asignada. Se creyó conveniente este formato de salida por su facilidad de uso posterior dentro del propio Matlab. La Figura 3.8 presenta un esquema con el formato del archivo de salida.

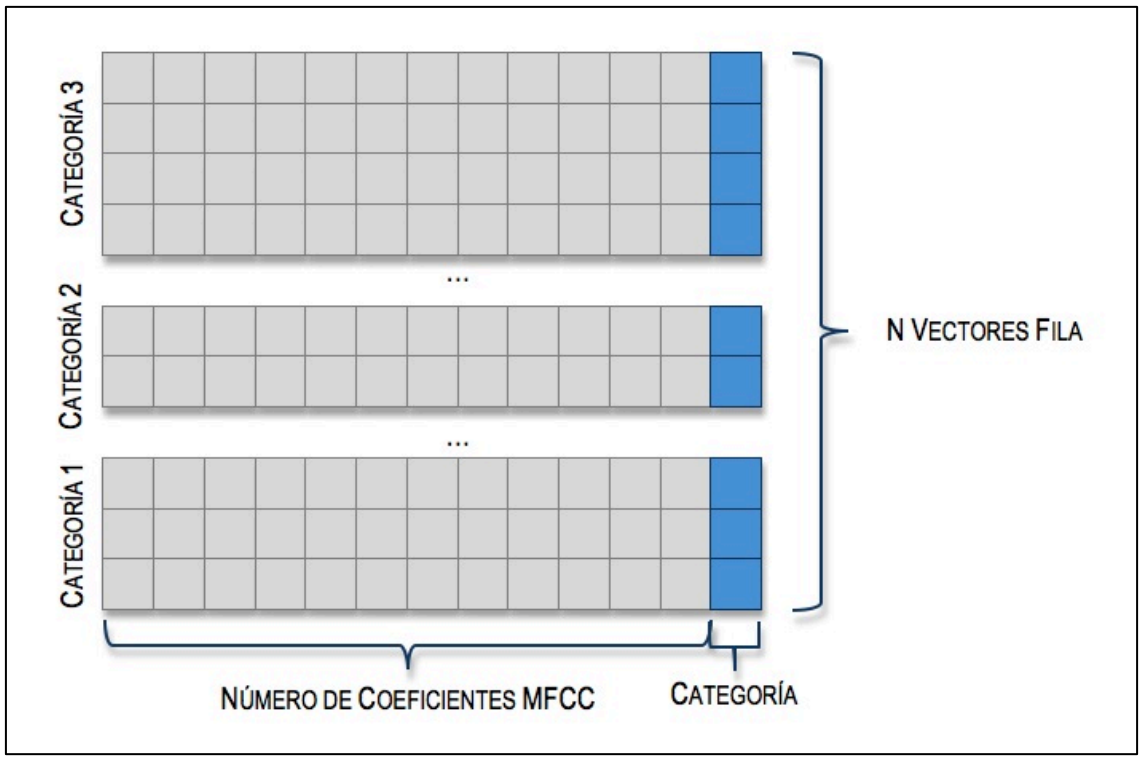

Figura 3.8. Formato del archivo de salida

3.3 Entrenamiento

Una vez obtenidos los datos de entrenamiento (vectores de coeficientes de Mel y etiquetas), damos el siguiente paso en nuestro sistema (ver Figura 2.1) y nos centramos en el bloque de integración. Como expusimos en la sección 2.2, tendremos en cuenta cuatro métodos para la integración de vectores de características: Media-Varianza, Media-Covarianza, DAR y MAR.

En todos los casos, el procedimiento seguido es el mismo, y sólo varía el método preciso utilizado. De forma general, el proceso es el siguiente:

1) Selección de la ventana.
2) Integración de la ventana.
3) Creación del vector integrado.
4) Si no se ha llegado al fin de la matriz de coeficientes, volver a 1) , en caso contrario se termina la ejecución.

Es importante reseñar dos aspectos: en primer lugar, en caso de que no haya muestras para completar la ventana, se sigue con el cálculo con el número de muestras disponibles; en segundo lugar, sólo se integran vectores que pertenecen a la misma categoría. Podemos ver el proceso de manera gráfica en la Figura 3.9.

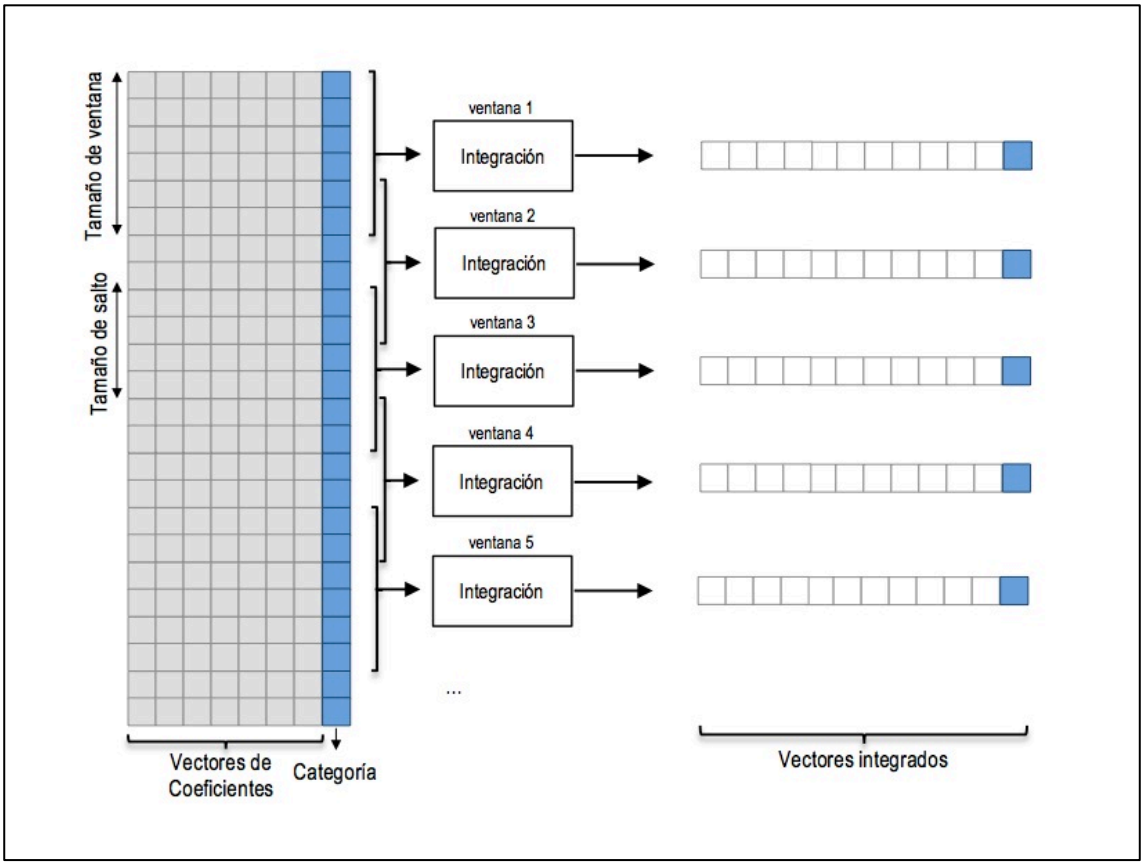

Figura 3.9. Esquema de integración de Matlab.

Como vemos en la Figura 3.9, el proceso de integración es el mismo independientemente de qué método se vaya utilizar, por lo que el único elemento diferencial en el esquema es el bloque de integración. A continuación detallaremos cómo se han implementado los distintos métodos.

Media-Varianza

Para realizar la integración bajo el método de la Media-Varianza simplemente utilizamos las funciones proporcionadas por Matlab para tal efecto, esto es, *mean(ventana)* y *var(ventana, 1)*, donde *ventana* se refiere a la ventana temporal que va a ser integrada.

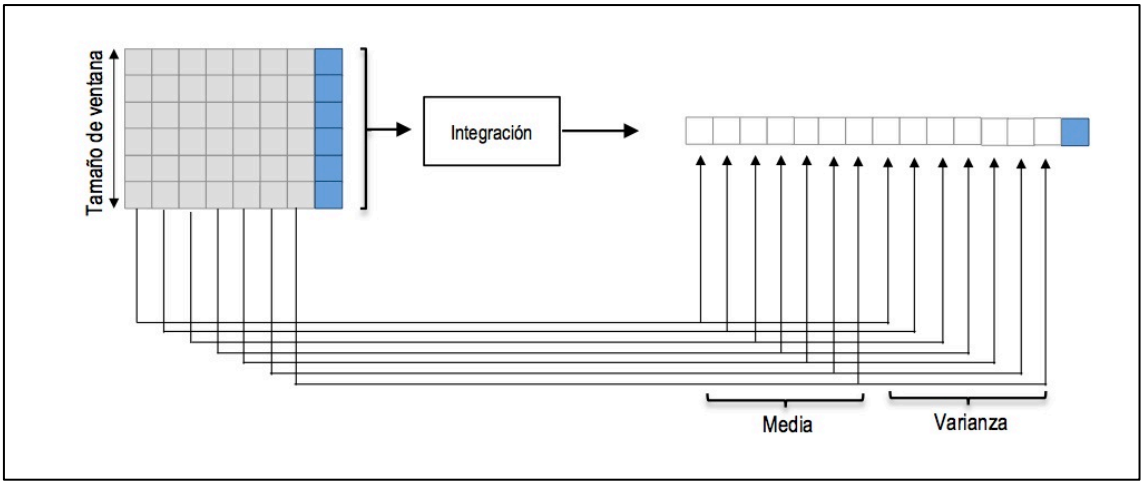

Figura 3.10. Esquema de la integración Media-Varianza

Media-Covarianza

Como se desprende de la Ecuación 2.12, el vector integración de este método consta de la media temporal de los coeficientes y de un vector formado apilando la triangular superior –incluyendo la diagonal- de la matriz de covarianzas.

De nuevo, el vector de medias se consigue a través de la función *mean* de Matlab, para conseguir la segunda parte del vector llamamos, en primer lugar, a la función *cov* , que devuelve la matriz de covarianzas y posteriormente, creamos el vector que acompaña al de medias. Ambas funciones reciben como argumento de entrada la ventana temporal.

DAR y MAR

Como mencionamos en el apartado 2.2.4 los métodos de integración basados en modelos autorregresivos multivariable son los más complejos que vamos a tratar. Para llevar a cabo su implementación en Matlab, nos apoyamos en un conjunto de funciones externas, llamado ARfit. ARfit es una colección de funciones de Matlab para el modelado y el análisis de series temporales multivariable con modelos autorregresivos.

En concreto, dentro de este conjunto de funciones, utilizamos *arfit*, que devuelve la estimación del modelo AR (autorregresivo) basado en el método de los mínimos cuadrados. Como parámetros de entrada, tenemos el rango de órdenes del modelo y la matriz de datos. Para una mejor búsqueda de parámetros hacemos que el orden mínimo y el máximo coincidan, *pmin* = *pmax*. La matriz de datos será, de nuevo, la ventana de coeficientes de Mel.

Esta función nos proporciona los parámetros requeridos por los métodos DAR y MAR para la integración de características temporales que, haciendo memoria, son los siguientes: las matrices $\{\boldsymbol{A}_1, \ldots, \boldsymbol{A}_P\}$ –que determinan cuánta información de las salidas anteriores está presente en la salida actual- y la media y la matriz de covarianza del ruido, v y $\boldsymbol{C}$, respectivamente.

Una vez conseguidos los parámetros, basta con formar el vector de integración tal y como formula la Ecuación 2.14, teniendo en cuenta las diferencias entre el método DAR y el MAR. Mientras que para el primero sólo tenemos en cuenta las diagonales de las matrices $\boldsymbol{A}_i$ y $\boldsymbol{C}$, para el segundo utilizamos las triangulares superiores completas, incluyendo las diagonales.

Entrenamiento

El siguiente paso, una vez se ha llevado a cabo la integración, es el entrenamiento del clasificador. Todos los aspectos relacionados con la clasificación se han implementado bajo el marco de la versión 3.1 de LIBSVM adaptada a Matlab. En concreto, se ha utilizado la función *svmtrain*, pasando como argumentos los vectores integrados de coeficientes de Mel, las etiquetas con las categorías y una cadena de texto con las opciones. El número de opciones es alto, proporcionando un gran abanico de posibilidades, en este proyecto se decidió iterar sobre el coste, la gamma y los pesos de cada clase. En el apartado siguiente se explicará con más detalle lo relacionado con los valores de los parámetros.

Por último, la función *svmtrain* devuelve un modelo, que es una estructura de datos que recoge los parámetros que definen el clasificador, entre ellos los vectores soporte. Este modelo será utilizado en el siguiente paso, las pruebas del clasificador.

Clasificación

La clasificación se lleva a cabo utilizando la función de LIBSVM *svmpredict*. Haciendo uso del modelo obtenido de la llamada a la función de entrenamiento (*svmtrain*), se realiza la llamada a *svmpredict*, junto los datos de test y sus etiquetas. A su salida, nos encontramos con las etiquetas de la clasificación, que nos servirán para evaluar el comportamiento del clasificador.

3.4 Selección de parámetros

Una vez introducidos los componentes que formarán parte de nuestro sistema, así como los parámetros que lo configuran, llega el momento de encontrar la combinación de valores que produce los mejores resultados.

Como hemos mencionado anteriormente, en esta fase de la implementación en Matlab se intentará encontrar un modelo que entregue unos resultados satisfactorios, si bien, el modelo final utilizado se buscará dentro de la misma aplicación de iOS. Por ello, en este momento, intentaremos establecer los parámetros relacionados con todos los bloques salvo el clasificador, es decir, los bloques de extracción de coeficientes de Mel y la integración. En la Figura 3.11 se muestra de forma esquemática los parámetros estudiados en este apartado.

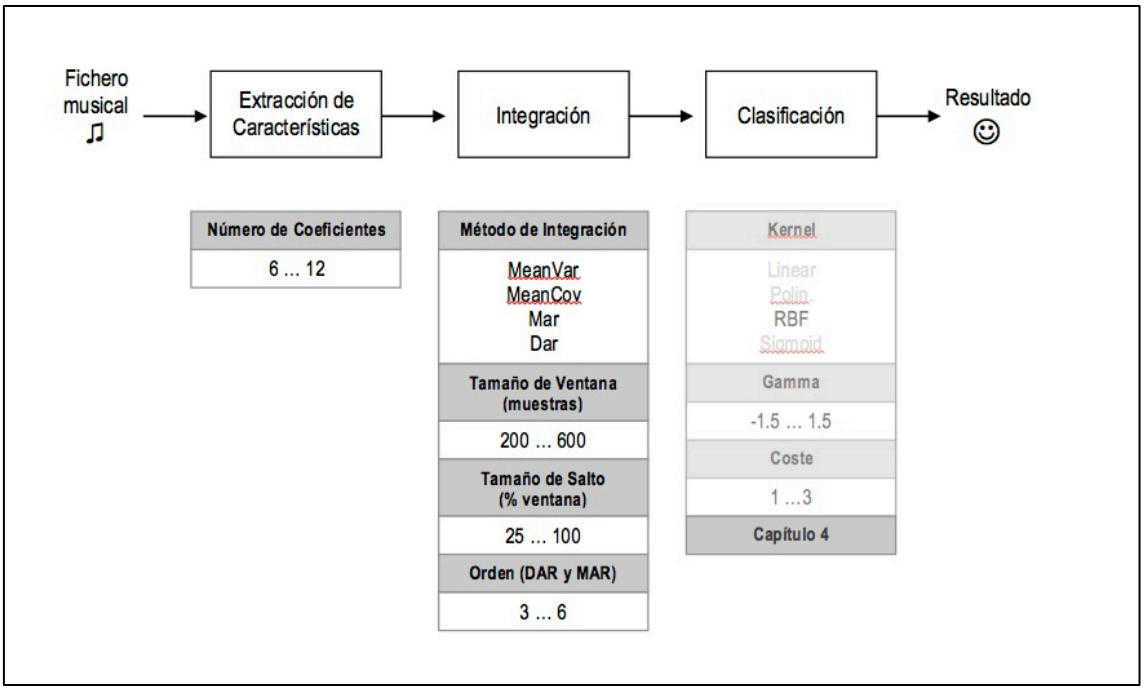

Figura 3.11. Esquema de los parámetros buscados en el Capítulo 3

A continuación, detallamos el estudio realizado para la búsqueda del valor de cada uno de los parámetros.

3.4.1 Coeficientes de Mel (MFCC)

El único parámetro estudiado para el cálculo de los coeficientes de Mel es el número de coeficientes. Como expusimos en 3.2.2.2, creímos conveniente utilizar una ventana Hanning de

16 ms, tal y como aparece en la mayoría de los estudios relacionados con el cálculo de los MFCC.

Para encontrar el número de coeficientes adecuado entre un rango de 5 a 12 –que es el rango comúnmente aceptado, se iteró entre los distintos tipos de integración, con un mismo tamaño de ventana y de salto, respectivamente 600 y 300 muestras. Se evaluaron los resultados según la probabilidad de acierto entregada al final del sistema. En la siguientes tablas se muestran los resultados obtenidos para los distintos métodos de integración. También se incluyen para qué coste y gamma se obtuvieron dichos resultados, conviene mencionar que los valores para esta columna se refieren al exponente sobre el que se eleva el 2, esto es, para un coste o gamma de -1, el valor real utilizado es de $2^{(-1)}$. Además, se han resaltado los mejores resultados obtenidos en cada método.

MEDIA-VARIANZA			
Número de Coeficientes	**Coste**	**Gamma**	**Tasa de acierto (%)**
5	-3	1	64,76
6	-5	2	63,32
7	-4	1	64,87
8	-4	1	64,30
9	3	5	64,35
10	2	5	64,61
11	0	5	64,46
12	0	5	64,64

Tabla 3.1. Probabilidad de acierto según el número de coeficientes de Mel para el método Media-Varianza

MEDIA-COVARIANZA			
Número de Coeficientes	**Coste**	**Gamma**	**Tasa de acierto (%)**
5	-4	1	67,93
6	-5	1	66,78
7	-4	1	65,76
8	-4	1	65,21
9	-1	4	64,70
10	-1	3	64,70
11	-2	3	64,70
12	-1	3	64,70

Tabla 3.2 Probabilidad de acierto según el número de coeficientes de Mel para el método Media-Covarianza

DAR			
Número de Coeficientes	**Coste**	**Gamma**	**Tasa de acierto (%)**
5	2	2	64,74
6	0	5	64,70
7	0	5	65,70
8	1	1	64,74
9	0	5	64,70
10	0	5	64,70
11	-4	0	64,72
12	0	5	64,70

Tabla 3.3. Probabilidad de acierto según el número de coeficientes de Mel para el método DAR

MAR			
Número de Coeficientes	**Coste**	**Gamma**	**Tasa de acierto (%)**
5	4	0	62,33
6	5	-1	63,80
7	5	-3	63,99
8	5	-4	63,97
9	5	-5	63,32
10	3	-4	61,94
11	4	-5	63,70
12	2	-2	63,50

Tabla 3.4. Probabilidad de acierto según el número de coeficientes de Mel para el método MAR

Para visualizar mejor los datos presentados en las tablas (Tabla 3.1-4), representamos de forma gráfica los resultados en la Figura 3.12.

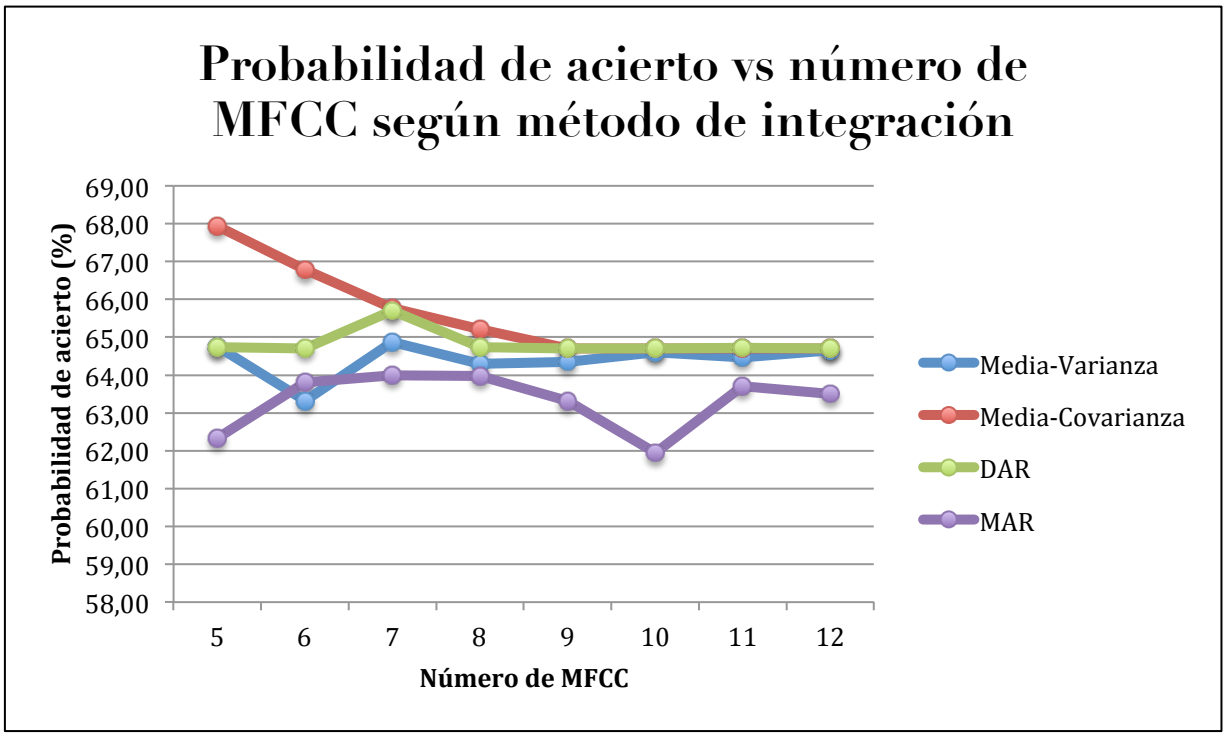

Figura 3.12. Evolución de la probabilidad de acierto para distinto número de coeficientes de Mel y método de integración.

A tenor de los resultados, podemos observar cómo, aunque el funcionamiento de todos los métodos y números de coeficientes es muy similar, para un número de coeficientes igual a 7 las características del sistema mejoran ligeramente. Es por esto, que decidimos utilizar 7 como el número de coeficientes de Mel usado en el resto del estudio.

3.4.2 Integración de coeficientes

3.4.2.1 Tamaño de ventana y tamaño de salto.

Estudiaremos de forma conjunta el tamaño de ventana y de salto más apropiados para el sistema. De nuevo, tendremos en cuenta el método de integración utilizado. Sin embargo, en esta ocasión, no iteraremos sobre el número de coeficientes usado, puesto que se decidió utilizar 7. A continuación se muestran los resultados obtenidos (Tabla 3.5).

	Tamaño de salto (% del tamaño de la ventana)			
Tamaño de ventana (muestras)	**25**	**50**	**75**	**100**
Media-Varianza				
200	63,35	67,36	65,61	64,59
300	65,53	68,64	67,26	65,75
400	67,22	69,76	68,85	65,88
500	69,61	71,16	67,97	66,37
600	70,54	71,32	69,57	67,38
Media-Covarianza				
200	59,38	60,23	59,50	58,84
300	59,99	60,53	59,71	58,38
400	59,89	60,59	59,75	58,74
500	59,93	60,69	59,53	59,07
600	59,81	60,87	59,41	59,08
DAR				
200	60,77	59,36	59,26	57,86
300	63,60	61,27	62,70	57,88
400	68,83	66,71	65,29	61,66
500	69,76	69,65	65,54	64,50
600	**70,29**	71,71	67,67	66,13
MAR				
200	70,85	70,18	65,81	61,19
300	71,55	75,69	68,33	64,33
400	73,65	79,26	70,27	62,90
500	74,73	70,99	61,27	57,01
600	75,55	76,87	61,65	52,95

Tabla 3.5. Probabilidad de acierto según el tamaño de ventana y de salto.

Tras un vistazo a la tabla de resultados (Tabla 3.5), podemos obtener varias conclusiones:

- El sistema mejora cuanto mayor es el tamaño de la ventana.
- El tamaño de salto idóneo se encuentra en el margen del 25-50%, a partir de ese valor, las prestaciones disminuyen.

- Un tamaño de salto más pequeño implica un mayor número de ventanas, lo que supone una mayor carga computacional.

Teniendo siempre presente que nuestro objetivo final es desarrollar una aplicación en un dispositivo móvil y teniendo en cuenta las conclusiones anteriores, se decidió utilizar una ventana de 600 muestras para la integración, con un tamaño de salto del 50% de la ventana, es decir, 300 muestras. Por consiguiente, de aquí en adelante, todos los cálculos y pruebas se realizarán utilizando 7 coeficientes de Mel e integrando sobre ventanas de 600 muestras con saltos de 300 muestras.

Una vez establecidos estos parámetros, continuamos nuestro estudio buscando el método de integración más conveniente.

3.4.2.2 Método de Integración

Tras la teoría expuesta en el apartado 2.2.3 de este documento, ha llegado el momento de decidir cuál de las diferentes opciones de integración (Media-Varianza, Media-Covarianza, DAR y MAR) se implementará en la aplicación. Para ello, realizaremos una serie de pruebas, teniendo en cuenta tres aspectos vitales: la probabilidad de acierto global, la probabilidad de acierto de cada clase y la complejidad computacional de cada uno de los métodos.

Para medir la probabilidad de acierto global, simplemente tenemos en cuenta el número de aciertos frente al número total de predicciones, expresando su cociente en tanto por ciento. Para obtener la probabilidad de acierto de cada clase hacemos uso de las llamadas matrices de confusión. La matriz de confusión es una herramienta de visualización empleada en aprendizaje supervisado, en la que cada fila de la matriz representa el número de predicciones de cada clase, mientras que cada columna representa a las instancias en la clase real. Las matrices de confusión son útiles para observar el comportamiento del clasificador en la predicción de las diferentes clases. Un ejemplo de matriz de confusión es:

Número de predicciones		**Clase real**		
		1	**2**	**3**
Clase predicha	**1**	34	0	4
	2	7	45	6
	3	3	3	23

Tabla 3.6. Ejemplo de matriz de confusión.

Los valores en la diagonal muestran los aciertos para cada clase, si sumamos los valores de una fila obtenemos el número de veces que se ha predicho una categoría. En cambio, si sumamos los valores de una columna, tendremos el número de veces que aparece un elemento de esa categoría en la realidad. Para obtener el porcentaje de acierto de una categoría *i* simplemente hacemos:

$$P(C_i) = \frac{Elemento(i,i)}{Suma\ de\ la\ fila\ i} \quad [3.3]$$

Tras esta breve introducción, pasamos a mostrar los resultados que se obtuvieron una vez ejecutadas las pruebas.

Método de Integración	Orden de Integración	Gamma	Coste	Prob. Acierto Total	Prob. Acierto Categoría 1	Prob. Acierto Categoría 2	Prob. Acierto Categoría 3
Mean-Var	N/A	2^(-1)	2^(-1)	**58,15%**	50,45%	57,41%	60,81%
Mean-Cov	N/A	2^(-2)	1	**55,42%**	50,45%	55,15%	56,91%
DAR	5	2^(-4)	2^(-1)	**53,85%**	40,54%	80,57%	0,3%
MAR	5	2^(-5)	2	**56,32%**	30,63%	84,87%	0,3%

Tabla 3.7. Resultados obtenidos para los diferentes métodos de integración, incluyendo: la probabilidad de acierto global y la probabilidad de acierto de cada categoría- Se incluye el valor de gamma y el coste por razones orientativas

Como adelantamos, también se llevó a cabo un estudio de la complejidad computacional de cada método de integración. A la teoría presentada en el apartado 2.2.4, añadimos ahora una prueba real ejecutada bajo el siguiente escenario:

- Datos de entrada: matriz de 31.319 x 8, donde cada una de las filas corresponde a un vector de 7 MFCC más un número con la categoría a la que pertenece.
- Se evalúa el tiempo de ejecución de cada una de las funciones mediante el par de funciones de Matlab *tic-toc*, que muestran el tiempo que transcurre entre la llamada a la primera y la llamada a la segunda.

Con este escenario de pruebas, los resultados obtenidos fueron clarificadores:

Método de Integración	Tiempo de ejecución (ms)
Media-Varianza	22,14
Media-Covarianza	44,53
DAR	214,4
MAR	246,1

Tabla 3.8. Tiempo de ejecución de cada método bajo el escenario propuesto.

Varios comentarios se pueden realizar tras observar los resultados obtenidos:

- Los métodos autorregresivos multivariable (MAR y DAR) sufren un problema grave de **balanceo**: la probabilidad de acierto para la categoría 3 no llega ni al 1%, mientras que, para la categoría 2, sobrepasa el 80%.
- Tanto para MAR como para DAR, el orden para el que se obtienen mejores resultados es 5.
- Tal y como se introdujo de forma teórica, los métodos no autorregresivos son mucho más rápidos.

A la vista de los resultados, creemos más que justificado que el método de integración en nuestra aplicación sea el de **Media-Varianza**, puesto que ha demostrado ser el más rápido y el que mejores resultados probabilísticos proporciona.

3.5 Conclusiones

En este capítulo se ha intentado explicar de una forma clara la etapa de desarrollo del proyecto bajo Matlab. Después de introducir la base de datos con la que trabajamos durante todo el proyecto, entramos de lleno al desarrollo del escenario de pruebas. En primer lugar, desarrollamos una interfaz para el etiquetado de canciones según unas categorías establecidas por el usuario, asociando dichas etiquetas a los vectores de coeficientes de Mel calculados para la canción en cuestión. Esta interfaz servirá de guía para el posterior desarrollo en iOS.

Más adelante, explicamos el método de entrenamiento y clasificación seguidos en Matlab, haciendo especial hincapié en la implementación de los distintos métodos de integración. Una vez expuestos todos los bloques que intervienen en el sistema, pasamos realizar la búsqueda de los parámetros que lo conforman, intentando escoger los más adecuado. Como resumen, los parámetros escogidos se muestran en la siguiente tabla:

Bloque	**Parámetro**	**Valor**
Extracción de MFCC	Número de coeficientes	7
Integración	Tamaño de ventana	600 muestras
	Tamaño de salto	300 muestras
	Método de Integración	Media-Varianza
	Orden	N/A

Clasificador	Gamma	Aún por definir
	Coste	Aún por definir

Tabla 3.9. Relación de parámetros seleccionados hasta el momento.

Es en este punto, con el modelo definido (salvo por clasificador), donde cerramos, al menos de momento, el desarrollo en Matlab. A partir de ahora, pasaremos a la siguiente etapa, que será crucial en el devenir del proyecto: el desarrollo en iOS.

4. Desarrollo de la aplicación para iOS

Tras haber llegado a formar un primer modelo en la etapa anterior, es el momento de afrontar el principal objetivo de este proyecto: desarrollar una aplicación para iOS que permita el etiquetado y clasificación de canciones en tiempo real.

El guión de este capítulo va a ser parecido en su estructura al anterior, aunque con más detalle. Primero, introduciremos algunos conceptos básicos de la programación en iOS, como el paradigma Modelo-Vista-Controlador (MVC). Seguidamente, expondremos la estructura básica de la aplicación, su esqueleto, para después explicar uno a uno cada uno de sus componentes principales. Una vez tratados todos los detalles relacionados con la programación será el momento de centrarnos en el problema que nos ocupa: implementar el etiquetador y el clasificador, para ello documentaremos qué *frameworks* y funciones se han utilizado y cómo se consiguió integrar el paquete de LIBSVM dentro de la aplicación. Por último, habiendo terminado el desarrollo de la aplicación, proseguiremos en la búsqueda del modelo óptimo, mediante el etiquetado y las pruebas en el simulador del dispositivo.

4.1 Conceptos básicos de la programación para iOS

A los conceptos introducidos en el capítulo 2 (apartado 2.4) sobre la programación en el marco de iOS, sumamos estos nuevos que ayudarán a comprender cómo desenvolverse mejor en el terreno.

4.1.1 Paradigma Modelo-Vista-Controlador

El diseño bajo Cocoa Touch se guía por un concepto llamado Modelo-Vista-Controlador (MVC), que es un procedimiento para dividir el código que forma una aplicación basada en interfaz de usuario. El modelo MVC divide todas las funcionalidades en tres categorías distintas:

- Modelo: Clases que contienen los datos de la aplicación.
- Vista: Construida a base de ventanas, controles y otros elementos que el usuario ve y con los que puede interactuar.
- Controlador: Conecta el modelo y la vista y contiene la lógica que decide cómo actuar ante las acciones del usuario.

El objetivo de MVC es hacer los objetos que implementan estos tres tipos de código tan distintos entre ellos como sea posible, cualquier objeto que se implemente debería ser rápidamente identificable como de una categoría u otra. MVC pretende ayudar a asegurar la reusabilidad del código. Una clase que implementa un botón genérico puede ser usado en cualquier aplicación. Sin embargo, un botón que realiza un cálculo específico debería ser utilizado para la aplicación para la que fue desarrollado.

Generalmente, cuando se crean aplicaciones en Cocoa Touch, se comienza implementando los componentes de la vista utilizando Interface Builder (visto en el capítulo 2). El controlador de la aplicación normalmente se compondrá de clases creadas por el usuario o por clases genéricas ya existentes, como los controladores incluidos en UIKit framework. Por supuesto, todas estas clases pueden ser extendidas en subclases, heredando los principales métodos y adquiriendo nuevos requeridos por el usuario

4.1.2 *Outlets* y Acciones

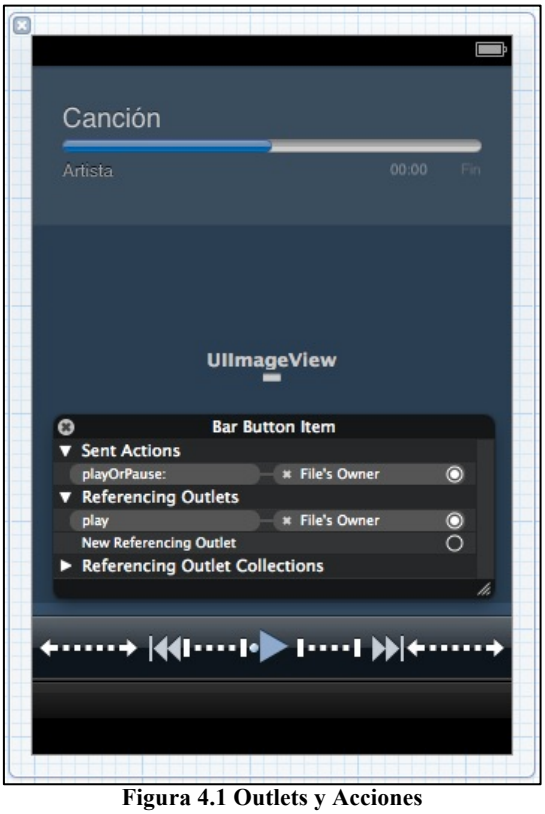

Figura 4.1 Outlets y Acciones

Las ***outlets*** o salidas en castellano son variables de clase con el único propósito de avisar a Interface Builder de que queremos conectar cierto objeto de la interfaz de usuario con cualquier objeto de nuestro código. Cualquier objeto que vaya a ser utilizado en la interfaz de usuario debería estar precedido de la palabra IBOutlet en su declaración. Cuando se abre Interface Builder, se escanean todas las cabeceras del proyecto para encontrar las declaraciones IBOutlet y de este modo permitir las interconexiones.

Las **acciones** son métodos parte de la clase controladora. Se declaran con la palabra IBAction, avisando a Interface Builder de que ese método puede ser activado tras la interacción con algún control. Normalmente, la declaración de un método de acción tendrá la siguiente forma:

```
(IBAction)nombreAccion:(id)sender;
```

donde `nombreAccion` es el nombre del método que, como cabía esperar, puede ser elegido por el usuario, siempre y cuando el tipo de retorno sea IBAction; por norma general, el argumento de entrada es de tipo id y su nombre suele ser `sender` (emisor en inglés). El control que activa la acción utilizará el argumento `sender` para autorreferenciarse. Por ejemplo, si tu método de acción fue activado tras pulsar un botón, el argumento `sender` contendrá la referencia del botón específico que fue pulsado.

En la Figura 4.1 vemos un ejemplo de cómo aparece en Xcode la relación entre el elemento de la interfaz, en este caso el botón de reproducción (*play*), su *outlet* y su acción. La declaración dentro del código de la aplicación del *outlet* y de la acción es, respectivamente:

```
@property (nonatomic, retain) IBOutlet UIBarButtonItem *play;
…
- (IBAction)playOrPause:(id)sender;
```

Para asociar al botón de la interfaz, el *outlet* llamado *play* y el método playOrPause, basta con dirigirse al fichero que contiene la interfaz de usuario (de tipo .xib) y hacer *click* derecho en

el elemento en cuestión. Aparecerá un menú flotante con distintas asociaciones posibles: "Sent Actions" (acciones que envía), "Referencing Outlets" (*outlets* a los que referencia) y "Referencing Outlet Collections" (colecciones de *outlets* a los que referencia).

Figura 4.2 Placeholders

Figura 4.3 Asignación de acciones o outlets

Para hacer efectivos los enlaces, simplemente clicamos en el círculo vacío (como el que aparece junto a New Referencing Outlet) y –manteniendo pulsado –llevamos la línea de unión que aparece hasta el apartado llamado "Placeholders" (Figura 4.2). Una vez allí, soltamos sobre "File's Owner" (que contiene los *outlets* y acciones declarados en el código) y aparecerá otro menú desplegable en el que se muestran los *outlets* o las acciones que pueden ser enlazadas (Figura 4.3).

4.1.3 *Application Delegate*

Cocoa Touch utiliza un tipo de clases llamadas *delegates* (delegadas, en castellano), que son las clases responsables de actuar en nombre de otro objeto. Las *application delegates* nos permiten hacer operaciones en ciertas ocasiones en nombre de la clase UIApplication. Cada aplicación iOS tiene una y sólo una instancia de la clase UIApplication , que forma parte de UIKit y es la responsable de la ejecución de la aplicación y controla las funciones a nivel de aplicación como encaminar las entradas a la clase controladora adecuada.

4.2 Estructura de la aplicación PFC

4.2.1 Esqueleto de la aplicación

Aunque es posible encontrar aplicaciones en iOS en el mercado que contengan una sola vista, es decir, toda la aplicación se encierra bajo una pantalla con controles, texto…, la mayoría de aplicaciones existentes contienen múltiples vistas. El ejemplo más sencillo de aplicaciones con múltiples vistas es una de utilidades. Una aplicación de utilidades se centra en una vista primaria pero ofrece una segunda vista que aumenta la funcionalidad de la primera o permite configurarla. Un ejemplo de este tipo es la aplicación de Bolsa incluida por Apple en iOS. Al pulsar sobre la *i* minúscula en la esquina inferior izquierda, pasamos de la vista primaria a la secundaria y el proceso contrario.

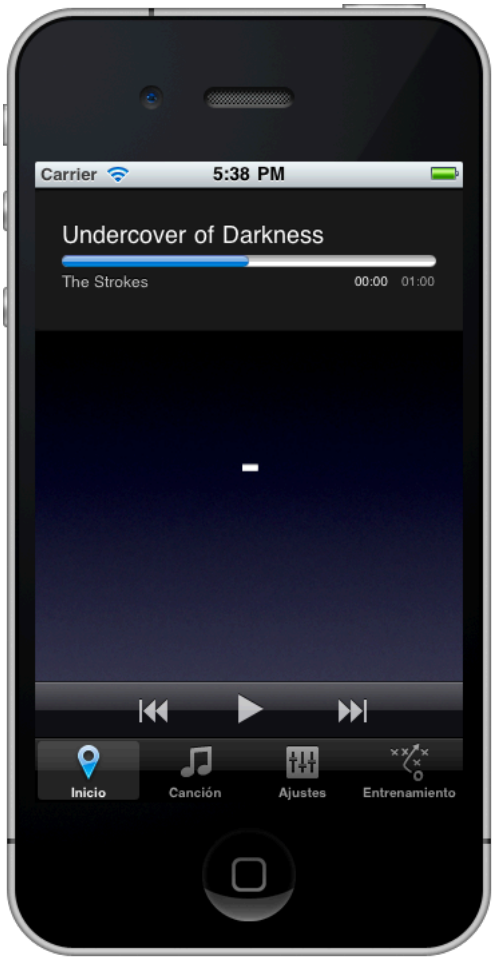

Figura 4.4 Pantalla de inicio con barra de pestañas.

Otro tipo de aplicación con múltiples vistas es la aplicación basada en navegación, que utiliza un controlador de navegación para presentar las vistas al usuario de forma jerárquica. Un ejemplo de este tipo de aplicaciones es la incluida de fábrica para el correo electrónico, Mail. En ella, el usuario puede navegar por sus cuentas de correo, pudiendo dirigirse hacia detrás o hacia delante pulsando sobre la barra de navegación superior.

Terminamos con las aplicaciones basadas en barra de pestañas, en ellas se muestra una barra con botones en la parte inferior de la pantalla. Pulsando sobre cualquiera de los botones, activamos el controlador de la vista del botón que acabamos de pulsar y se muestra, por tanto, dicha vista. Un ejemplo de este tipo de aplicaciones es el gestor de contactos de iOS.

Es importante destacar, que es posible combinar los distintos tipos, como por ejemplo ocurre en la aplicación iPod de los dispositivos iOS, donde se utiliza una barra de pestañas a la vez que la navegación.

Se decidió, para nuestro proyecto, utilizar una aplicación basada en barra de pestañas, pues proporcionaba una interfaz limpia y fácil de utilizar para el usuario. Además, la naturaleza de la aplicación que se pretendía diseñar no contaba con niveles jerárquicos que aconsejaran utilizar una aplicación basada en navegación.

En la Figura 4.4 se muestra una captura de la pantalla de inicio de la aplicación. Lo más importante en este momento es fijarnos en la barra inferior, la barra de pestañas. En ella podemos ver cuatro iconos diferentes, tantos como vistas tendrá el programa: Inicio, Canción, Ajustes y Entrenamiento.

En realidad, en esta captura de pantalla existen dos vistas: la barra de pestañas y la vista de Inicio. La *application delegate*, llamada PickersAppDelegate (la nomenclatura *Pickers* proviene de un proyecto de prueba creado con la ayuda de [21] que, junto con el sitio web [22] y la API de Apple conformaron casi la totalidad de las referencias sobre la programación en

iOS), se encarga –entre otras cosas –de controlar qué vista se muestra en cada momento teniendo en cuenta la interacción del usuario con la barra de pestañas. Para ello, se ha añadido programáticamente una instancia de la clase UITabBarController a PickersAppDelegate y, posteriormente, se ha enlazado esta barra de pestañas a un *outlet* creado para tal efecto en Interface Builder. La clase UITabBarController se encarga de gestionar los eventos recibidos por la barra de pestañas, sin embargo, delega sus funciones a PickersAppDelegate.

Una vez creada la interacción básica entre la barra y la aplicación se personalizaron los elementos de la barra, tanto en su nombre como en la imagen a mostrar. A grandes rasgos, puesto que se explica más adelante, la vista "Inicio" se encarga de reproducir la música y mostrar las emociones –mediante imagen y/o texto –que el clasificador determina. "Canción" permite seleccionar qué canción vamos a clasificar o etiquetar. En "Ajustes" podremos configurar el comportamiento de la aplicación así como crear nuestro propio clasificador. Por último, en la vista de "Entrenamiento" podremos etiquetar y entrenar el clasificador.

4.2.2 Componentes básicos

Figura 4.5. Elementos del proyecto PFC en XCode

Toda aplicación de iOS utiliza los siguientes ingredientes (Figura 4.5): clases, otras fuentes, recursos y *frameworks* para crear el producto final, el archivo de aplicación (de extensión .app). En este sub-apartado estudiaremos cómo es cada uno de estos ingredientes.

4.2.1.1 Clases y otras fuentes

En la carpeta de clases encontramos todas las clases escritas en Objective-C, C y C++. Se pueden crear subcarpetas para una visualización más apropiada. Si echamos un vistazo a nuestro proyecto (ver Figura 4.6) podemos encontrar clases y cabeceras escritas en los distintos lenguajes mencionados arriba. Por su funcionalidad distinguimos cuatro tipos de elementos:

1) Una clase AppDelegate, de la que hablamos anteriormente.

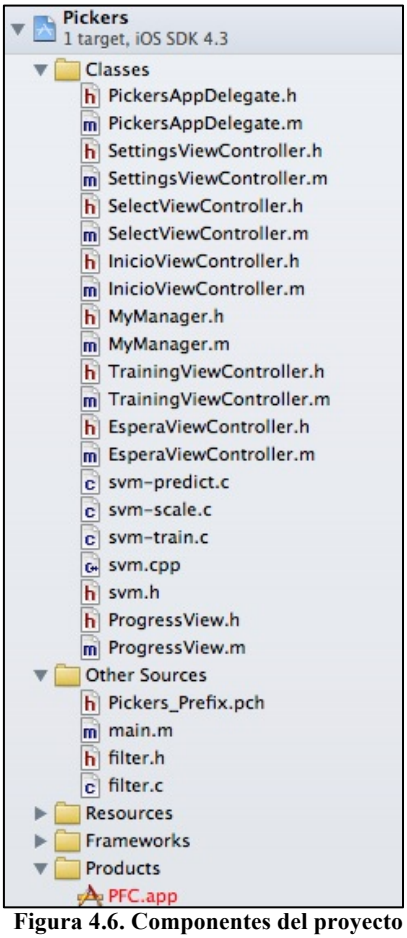

Figura 4.6. Componentes del proyecto en Xcode

2) Varias clases ViewController, que son los controladores de las diferentes vistas.

3) Una clase de compartición de recursos, MyManager.

4) Clases implementadas en C y C++ que corresponden con la integración de LIBSVM en la aplicación.

Detallaremos su funcionalidad en el apartado 4.3. En la carpeta "Other Sources" u Otras Fuentes, encontramos normalmente ficheros no escritos en Objective-C, si bien, nada impide que puedan ser incluidos en la carpeta de Clases, tal y como hemos comprobado anteriormente.

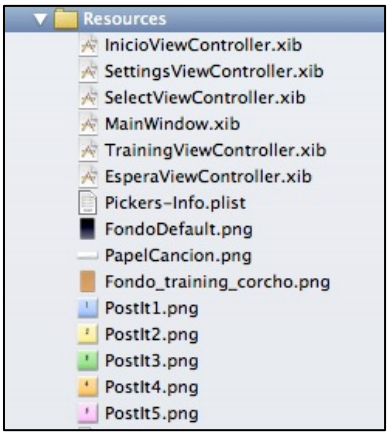

Figura 4.7. Contenido de la carpeta Resources.

4.2.1.2 Recursos

'Resources' o Recursos sirve de contenedor de cualquier tipo de fichero que no sea de código. Se pueden incluir imágenes, sonidos, archivos de texto… cualquier tipo de recurso que sea útil para el desarrollo de la aplicación.

Además, en esta carpeta deberíamos encontrarnos obligatoriamente con los siguientes objetos:

- Las interfaces de las vistas: Los elementos con extensión .xib guardan información sobre las interfaces gráficas de usuario de nuestro proyecto. Podemos ver en la Figura 4.7 como, en efecto, contamos con interfaces para "Inicio", "Ajustes" o "Settings", Canción o "Select" y Entrenamiento o "Training".
- La interfaz de la ventana principal de la aplicación, MainWindow.xib.

- La lista de propiedades que contiene información sobre nuestro programa, Pickers-Info.plist, que se genera automáticamente.

4.2.1.3 Frameworks

En el capítulo 2, concretamente en la sección 2.4.2.1, introdujimos el concepto de *framework* y enunciamos la mayoría de ellos. En este punto, concretamos los *frameworks* utilizados para desarrollar la aplicación. En la Tabla 4.1 aparecen tales *frameworks*, cualquier *framework* o librería incluida dentro de la carpeta *Frameworks* se enlazará con la aplicación y posibilitará el uso de objetos, funciones y recursos contenidos dentro de ellos.

Capa de iOS	Framework	Descripción
COCOA TOUCH	UIKit	Proporciona la infraestructura clave para implementar aplicaciones gráficas orientadas a eventos en iOS. Está presente en todas las aplicaciones.
MEDIA	AV Foundation	Captura y reproducción de vídeo y audio.
	Core Audio	Manipulación de audio. Incluye AudioToolbox
	Quartz Core	Contiene Core Animation, que es una tecnología de animación avanzada y composición.
	Core Graphics	Interfaz para la creación de gráficos
CORE SERVICES	Foundation	Proporciona soporte a Core Foundation Framework
CORE OS	Accelerate	Interfaces para cálculos matemáticos complejos y DSP

Tabla 4.1. Frameworks utilizados en la aplicación PFC, ordenados de forma descendente por capa de iOS.

4.2.1.4 Producto

La carpeta Products o Producto contiene la aplicación que el proyecto produce cuando se compila. Al expandir Products aparece un elemento con extensión .app llamado PFC.app que es la aplicación que estamos desarrollando.

4.3 Interfaz de usuario

En este apartado explicaremos las diferentes pantallas que se encontrará el usuario para interactuar con la aplicación. Como se introdujo en el apartado anterior, la aplicación está basada en un sistema de navegación por pestañas, donde una y sólo una puede ser visible en el momento. Se consideró que la interfaz debía ser lo más fácil e intuitiva posible, por ello se creyó conveniente que las siguientes cuatro pestañas o pantallas eran más que suficientes para abarcar toda la funcionalidad del programa.

Adicionalmente, se hizo un especial esfuerzo en realizar una aplicación que llamase en cierta medida la atención del usuario. Gracias al uso de animaciones, se intentó dar un aspecto más dinámico y fresco. Consideramos que, en el terreno de las interfaces gráficas de usuario, las animaciones juegan un papel sumamente importante y que, en ocasiones, no es tomado con importancia. Siendo la plataforma intermedia entre el usuario y el programa, la interfaz debe ser, no sólo fácil e intuitiva, sino también atractiva. Es por ello que hicimos especial hincapié en incluir animaciones en una medida justa.

En iOS, las animaciones siguen la estructura que se muestra en la siguiente figura.

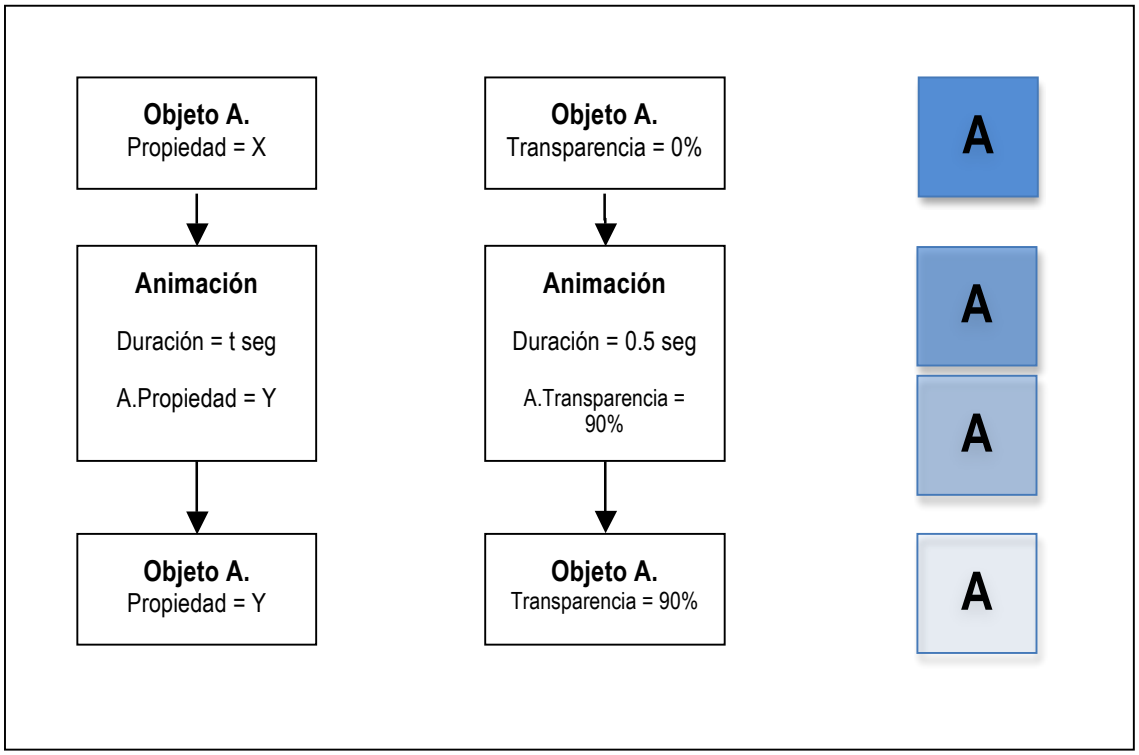

Figura 4.8. Esquema de animaciones en iOS.

Para configurar las animaciones utilizaremos los siguientes comandos:

`[UIView beginAnimations:nil context:nil]`	Marca el inicio de un bloque de animaciones.
`[UIView setAnimationDuration:0.6]`	Determina la duración, en segundos, de la animación.
`[UIView commitAnimations]`	Marca el fin del bloque de animación y ordena su ejecución

Tabla 4.2. Comandos habituales para la ejecución de animaciones en iOS.

A lo largo de los siguientes subapartados, aparecerán las distintas animaciones utilizadas para crear el efecto visual deseado.

Comencemos pues con el estudio de las distintas interfaces implementadas en la aplicación PFC.

4.3.1 Inicio

Es la primera pantalla que ve el usuario al arrancar el programa (ver Figura 4.9). Su objetivo es la reproducción de los archivos musicales mientras se muestra a qué categoría pertenece el fragmento que se está escuchando. Esta interfaz tiene, por tanto, dos funciones básicas: la reproducción de música y la clasificación. Dentro de la clasificación, tenemos en cuenta los bloques de extracción de MFCC, la integración de los mismos y la clasificación propiamente dicha.

Para la reproducción de música se utiliza un objeto de la clase AVAudioPlayer, perteneciente a AVFoundation Framework, que proporciona la reproducción de archivos de audio (archivos *.wav* en este caso). El resto de bloques –extracción de MFCC, integración y clasificación –se estudiarán en el siguiente apartado.

Teniendo siempre en cuenta al usuario, se consideró oportuno que nada más arrancar la aplicación, se clasificase una canción para que, de este modo, el usuario pudiera empezar a escuchar música simplemente pulsando el botón de *play*. La clasificación de esta primera canción se hace bajo el modelo de clasificación por defecto, esto es, el modelo que clasifica en "Alegre"-"Normal"-"Triste". Dicho esto, el usuario puede cambiar de canción ya sea mediante el botón de *flash-forward* o *rewind*, o mediante la interfaz "Canción". En la Figura 4.9 mostramos de qué componentes está formada la interfaz "Inicio".

Figura 4.9. Interfaz “Inicio” y componentes

1) **Título de canción**. Muestra el título de la canción que se está reproduciendo o que está cargada para ser reproducida.
2) **Barra de progreso**. Muestra de forma gráfica el tiempo que se ha reproducido la canción.
3) **Nombre del artista/Título del álbum**. Muestra tanto el nombre del artista como el título. Gracias al uso de un temporizador, podemos alternar entre estos dos campos de tal manera que no sean necesarios dos etiquetas de texto sino solamente una. Además, la transición entre ambos valores se realizó de forma gráfica mediante el uso de las animaciones. En este caso, se utilizó una disolución. El efecto de la disolución se consigue de la siguiente manera:
    a. Se configura la transparencia de la etiqueta al 100%, se ejecuta el cambio de forma gradual.
    b. Se cambia el valor de la etiqueta, de “Artista” a “Álbum”, por ejemplo.
    c. Se configura la transparencia de la etiqueta de 100 a 0%, gradualmente.

Este efecto es muy fácil de configurar, pero confiere a la aplicación un aspecto más elegante y menos aburrido.

4) **Marcador de tiempo/Duración**. En este caso, creímos conveniente mostrar en todo momento tanto el momento actual de reproducción como la duración de la pieza musical.

5) **Imagen de la categoría**. En el fondo de la interfaz se sitúa una imagen, predefinida por categoría, y que cambia a la par que cambia la categoría del fragmento que estamos escuchando. De nuevo, se hace uso de una transición de disolución, para crear un efecto visual agradable, de manera que la imagen no "simplemente aparezca". Las imágenes –de formato PNG, aunque admite otros –se incluyen en la carpeta de "Resources" del proyecto y, por supuesto, pueden ser cambiadas

6) **Número/Nombre de la categoría**. Muestra en pantalla el número o nombre de la categoría. Para incluir el nombre de la categoría es necesario dirigirse a la pestaña de "Entrenamiento" e introducir allí por teclado el nombre deseado. Tanto la imagen como el nombre/número de la categoría cambian al unísono con la misma transición, de forma gradual, de forma agradable para la vista.

7) **Controles de reproducción**. Es una barra de herramientas que contiene los botones básicos para la reproducción de música (de izquierda a derecha en la Figura 4.9): *rewind* (rebobinar), *play* (reproducir) y *fast forward* (avance rápido). Una llamada a *rewind* o *fast forward* no cambia simplemente de canción, sino que además llama secuencialmente a las funciones de extracción de MFCC, integración y clasificación, para que el usuario no tenga que ocuparse de ningún detalle de bajo nivel.

4.3.2. Selección

Esta interfaz muestra la lista de canciones que el usuario puede etiquetar o reproducir. En realidad, es una solución temporal a un problema que se presentó desde que se dieron los primeros pasos del proyecto: el propietario del sistema operativo y desarrollador de las APIs, Apple, no permite el uso libre de las canciones guardadas en la biblioteca musical del dispositivo. Permite la reproducción e incluso la ejecución de filtros predefinidos, pero no permite obtener las muestras PCM de las canciones para su posterior uso. Este hecho se ha mantenido durante todas las versiones del sistema operativo, desde la 1 hasta la actual, la 4. Para saltarnos de alguna manera esta limitación, consideramos que era una buena solución incluir como recurso un conjunto de canciones, en este caso, aquéllas de la base de datos.

Una vez explicado este inconveniente, comencemos con el estudio de esta interfaz. Se trata de una vista basada en una tabla, que es su único componente. Las tablas se implementan creando un objeto de la clase UITableView, a su vez éstas contienen objetos de la clase UITableViewCell, que se refiere a cada elemento –cada fila –de la tabla. Es posible definir el aspecto que tendrá la tabla y sus filas. Nuestra tabla tendrá el aspecto mostrado en la Figura 4.11, donde se muestran los nombres de las canciones acompañadas por el nombre del grupo autor.

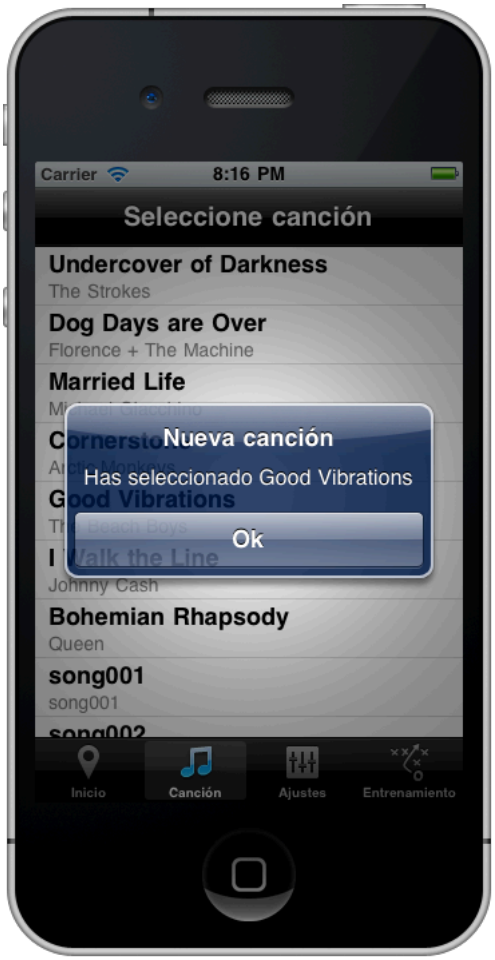

Figura 4.10. Mensaje de confirmación al tras canción.

Una vez definido el formato de la tabla, resta proporcionar a la tabla la fuente de los datos, para ello se hace uso de la clase NSArray. En la aplicación se utilizan dos arrays para mostrar los datos: uno para la canción y otro para el artista. Una vez el usuario ha pulsado sobre el elemento deseado, el controlador de la tabla buscarán en ambas tablas qué elemento ha sido escogido con ayuda del índice de la fila pulsada. Cuando la canción ha sido seleccionada, el sistema muestra por pantalla un mensaje de confirmación (ver Figura 4.10). Desde ese momento, se establece que la canción que se va a reproducir o a etiquetar es la seleccionada.

Es importante mencionar que el orden tanto de reproducción como de etiquetado es descendente, esto es, una vez escogida la canción, por ejemplo, número 7, la siguiente en

aparecer de forma automática será la 8, a no ser que el usuario prefiera elegir otra. Se decidió este proceder por su sencillez.

Finalmente, los elementos que forman parte de la interfaz "Canción" son los que se muestran en la siguiente figura.

Figura 4.11. Elementos de la interfaz "Canción"

1) **Título**. Es el título de la vista, es orientativo para el usuario.
2) **Título de la canción** y 3) **Nombre de artista**, se muestran dentro de la misma fila de tabla, pero con el nombre del artista con un tamaño menor de fuente y un color más claro, dando más protagonismo al nombre de la canción.

De nuevo, se hizo uso de efectos de animación para dar a la interfaz un aspecto más dinámico. En este caso, una vez pulsado el botón "Canción" en la barra de pestañas, la lista de canciones completa aparece de abajo hacia arriba, moviendo toda la vista para tal efecto.

4.3.3 Ajustes

En la pantalla de "Ajustes" –tercera pestaña de la barra –se recogen aquellos aspectos que se consideraron objeto de ser configurados por el usuario, ya sean relativos al reproductor o al clasificador.

Como hemos mencionado, el programa arranca con un clasificador por defecto y que, si no se desea lo contrario, será el que funcione durante la ejecución del programa. Si se decide usar un clasificador personalizado, esta interfaz es el lugar para iniciarlo. Para tal efecto, es necesario, en primer lugar, encender el interruptor "Personalizar", que por defecto está apagado. Tras ello, el usuario puede establecer ciertos aspectos del clasificador, como el número de categorías, el coste, la gamma y el *kernel*. Para la selección del *kernel* se decidió crear otra vista de apoyo (ver Figura 4.13), basada en la clase UIPickerView, que permite al usuario elegir un valor de entre una selección. Una vez establecidos los parámetros del clasificador, el usuario puede comenzar a entrenarlo.

En la Figura 4.12 se muestran los elementos de la interfaz "Ajustes". Nótese que en esta ocasión la captura de pantalla no se ha realizado del simulador de iOS, esto es debido a que la vista que contiene los ajustes es una vista que permite el *scroll* vertical (o desplazamiento vertical), esto es, la vista es más alta que la altura de la pantalla del iPhone.

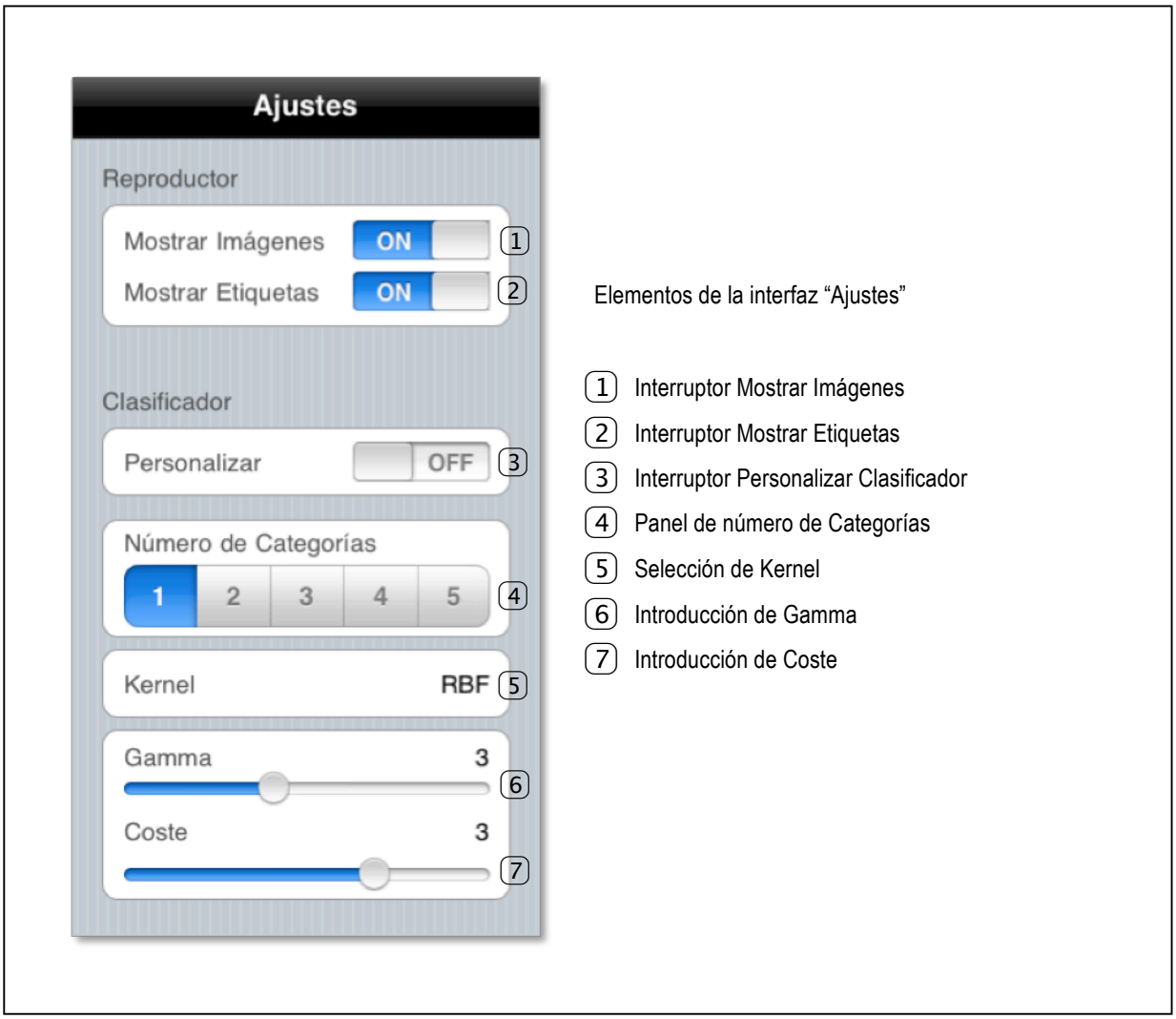

Figura 4.12. Elementos de la interfaz "Ajustes"

1) **Interruptor Mostrar Imágenes**. Activa o desactiva el visualizado de imágenes en la pantalla "Inicio" acordes con la categoría que se está reproduciendo.
2) **Interruptor Mostrar Etiquetas**. Activa o desactiva el visualizado de etiquetas en la pantalla "Inicio" acordes con la categoría que se está reproduciendo. Es ideal en el caso de que se prefiera ver solamente las imágenes de la categoría.
3) **Interruptor Personalizar Clasificador**. Cuando está activado, el programa utiliza el clasificador personalizado del usuario, cuando no, utiliza el clasificador por defecto.
4) **Selección de *kernel***, cuando es pulsado aparece la vista de selección de *kernel* (Figura 4.13), una vez elegido, es necesario pulsar el botón "Aceptar" para volver a la vista de Ajustes.
5) **Introducción de Gamma**. Este *slider* (o deslizador) numérico permite establecer el parámetro gamma del clasificador. Es importante hacer notar que el número mostrado

corresponde con el exponente al que se elevará el 2, para proporcionar el valor real de gamma al clasificador.

6) **Introducción de Coste**. Similar a 5) pero con el parámetro de coste.

Figura 4.13. Vista de selección de kernel.

4.3.4 Entrenamiento

La pantalla de entrenamiento es una de las más importantes de toda la aplicación. En ella, el usuario etiquetará las canciones mientras las escucha para posteriormente, entrenar el clasificador con los resultados del etiquetado. Se intentó que la interfaz fuera lo más atractiva visualmente posible. Tras varios diseños, se consideró que el conjunto formado por unos *post-its* sobre un corcho daba una imagen acorde con el hecho de entrenamiento. Además, en esta ocasión, la barra donde se muestra la información sobre la canción actual está situada sobre un papel rasgado, dando un aspecto aún más fresco. Como último detalle de diseño, la primera vez que se entra en la interfaz de "Entrenamiento", la hoja de papel se desliza, descendiendo lentamente, hasta alcanzar su posición final. Como contrapartida, la barra de controles de reproducción, se desliza, pero de forma ascendente. Por su parte, los *post-its* aparecen de forma gradual. Se consideró adecuado que estos efectos sólo se ejecutaran una vez porque era posible que tras varias ejecuciones, pudieran cansar al usuario.

Detallemos los elementos de la interfaz con ayuda de la Figura 4.14.

Figura 4.14. Elementos de la interfaz Entrenamiento

1) **Título de la canción**. Muestra el nombre la canción que se está etiquetando.
2) **Barra de progreso**. Muestra de forma gráfica el tiempo que se ha reproducido la canción.
3) **Nombre del artista/Título del álbum**. Idéntico al de la interfaz de "Inicio" (apartado 4.3.1).
4) **Marcador de tiempo/Duración**. Idéntico al de la interfaz de "Inicio" (apartado 4.3.1).
5) **Botón de entrenamiento**. Una vez etiquetada la canción, se entrena al clasificador con los nuevos datos, tras pulsar este botón.
6) **Botones de etiquetado**. Permiten al usuario decidir a qué categoría pertenece el fragmento que acaba de escuchar. Muestran el número de categoría, aunque también es posible añadir una etiqueta de texto, para hacer más fácil el etiquetado al usuario.
7) **Controles de reproducción**. Idénticos a los de la interfaz de "Inicio" (apartado 4.3.1).

Tras explicar los detalles concernientes a la interfaz y conociendo los elementos de la misma, pasamos a explicar cómo es el proceso de etiquetado y posterior entrenamiento de una canción. El punto de partida es el arranque de la aplicación, con el clasificador por defecto y con la primera canción de la lista.

1) Si se desea cambiar de canción, nos dirigimos a la interfaz "Canción".
2) Si se desea cambiar de modelo de clasificación, nos dirigimos a "Ajustes".
3) Nos dirigimos a la interfaz de "Entrenamiento".
4) Si deseamos añadir texto a los botones de etiquetado, pulsamos sobre el texto por defecto "Categoría x", donde x= 1,2,…,5. Aparecerá el teclado incluido en el SO e introduciremos el nombre deseado (ver Figura 4.15a). Como aclaración, si el número de categorías es menor que 5, se deshabilitarán los botones correspondientes a las últimas categorías no utilizadas.
5) Comenzamos a reproducir la canción, pulsando el botón *play*. Comienza el etiquetado.
6) Cuando el usuario crea identificar un segmento de la canción como perteneciente a alguna categoría, simplemente pulsa el botón de dicha categoría cuando crea que ha dejado de pertenecer a dicha categoría.
7) Repetir 6) hasta que se alcance el final de la canción. Una vez acabada, determinar a qué categoría pertenecía el último fragmento pulsando el botón correspondiente.
8) Cuando el etiquetado ha finalizado, aparecerá un mensaje de alerta (ver Figura 4.15b) anunciando el fin del etiquetado.

9) Si el usuario está conforme con el etiquetado, pulsa el botón de entrenamiento, incluyendo las nuevas etiquetas al conjunto de etiquetas de entrenamiento presente. Si cree conveniente volver a etiquetar, basta con volver a 5).

Figura 4.15. A la izquierda, introducción del nombre de categoría(a). A la derecha, mensaje de fin de etiquetado (b).

Los detalles de programación relacionados con los procesos cruciales incluidos en la aplicación se explican en el siguiente apartado. Se han considerado como cruciales la extracción de MFCC, la integración de dichos coeficientes y el entrenamiento y la clasificación.

4.4 Implementación del sistema de clasificación

En este apartado llevaremos a cabo un estudio más exhaustivo de las funciones principales de la aplicación. Como la teoría sobre la que se sustentan estas funciones fue explicada a lo largo del capítulo 2, nos centraremos en destacar los aspectos más técnicos.

4.4.1 Extracción de MFCC

El mayor reto a la hora de extraer los coeficientes de Mel era encontrar los mejores métodos para llevar a cabo cálculos complejos. Teniendo siempre en cuenta que estamos tratando con una aplicación diseñada para plataformas móviles –recursos limitados –debíamos encontrar la manera más rápida y menos costosa de hacer las cosas.

Afortunadamente, dentro del *framework* Accelerate, nos encontramos con la API vDSP, que proporciona funciones matemáticas para distintas aplicaciones como el tratamiento de voz, sonido, video e imágenes. Las funciones de vDSP operan con datos reales y complejos e incluyen métodos para conversión de tipos de datos, FFT (Fast Fourier Transform o Transformada Rápida de Fourier), y operaciones vector-vector y vector-escalar. Con esta funciones incluidas por el fabricante, conseguimos sacar el mayor rendimiento para las operaciones que, de otro modo, podían malograr el rendimiento de la aplicación.

Aparte de las funciones incluidas en vDSP, para aquellos cálculos más sencillos, simplemente nos apoyaremos en las funciones propias de C. Recordemos que el compilador acepta órdenes de C, C++ y Objective-C indistintamente.

Recordemos el esquema presentado en la Figura 2.3 y que, por comodidad, reproducimos en la Figura 4.16. En este apartado explicaremos los aspectos más importantes de cada una de las etapas del cálculo de los coeficientes de Mel, la mayoría de ellos implementados dentro la función *mfcc*.

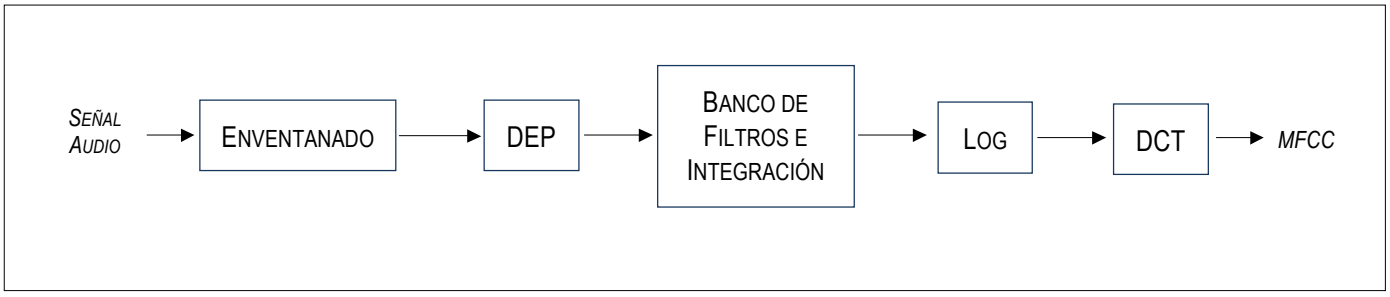

Figura 4.16. Esquema de extracción de los coeficientes de Mel.

4.4.1.1 Señal de audio

El primer paso, paso preliminar, se sitúa fuera de la función de cálculo de los coeficientes. Se trata del método *cargarCancion*, `(void)cargarCancion:(NSString *)tema` , que recibe como entrada una cadena de caracteres con el nombre del fichero de audio que se va a tratar. Este método tiene dos fines distintos: por un lado, inicia el reproductor de música,

como ya vimos y, por otro, obtiene las muestras PCM de los archivos musicales. Nos centraremos en este último aspecto.

Para obtener las muestras de un fichero de música debemos completar dos requisitos: crear una descripción del audio que vamos a tratar y crear una plataforma para tratarlo. Veamos el procedimiento que hemos seguido:

1) Abrimos el fichero de audio y lo volcamos a una nueva estructura ExtAudioFileRef, que nos ayudará a tratarlo.
2) Especicficamos las propiedades del audio que vamos a tratar, para ello creamos un objeto de la clase AudioStreamBasicDescription, donde, entre otros parámetros, configuraremos la tasa de muestreo a 16 kHz, el número de canales a 1 y el formato a PCM lineal.
3) Aplicamos el formato configurado al objeto ExtAudioFileRef creado en 1).
4) Creamos un buffer de audio, AudioBufferList. De nuevo, declaramos el número de canales que tendrá nuestro audio.
5) A través de la función ExtAudioFileRead, leemos el archivo de audio y lo guardamos en el buffer creado a tal efecto. Esta función devuelve un código de resultado, de error si no se ha podido completar la operación. Como parámetros de entrada tiene la descripción del audio creada en 2), y la dirección donde guardar el número de muestras leídas, así como del buffer donde se guardarán.
6) Finalmente, se convierten las muestras leídas a float. Nuestro fichero .wav ya está listo.

4.4.1.2 Enventanado

Tras obtener las muestras de audio, llamamos a la función del cálculo de coeficientes de mel, `(void)mfcc:(uint)win:(uint)filters:(uint)coef` , que recibe como parámetros el tamaño de la ventana, el número de filtros utilizado y el número de coeficientes de resultado, respectivamente. Como obtuvimos en el capítulo 3, utilizaremos 256 muestras de ventana, 29 filtros y 7 coeficientes de Mel.

Para el enventanado, traspasamos los datos del buffer a un *array* de dimensiones *Nº de ventanas* x *Tamaño de ventana*, donde el número de ventanas obedece a la Ecuación 3.1.

4.4.1.3 DEP

Es en este momento en el que comenzamos a utilizar las funciones incluidas en vDSP. El primer paso es la conversión de tipos. Convertiremos nuestras muestras de tipo float en objetos DSPSplitComplex, que representa números complejos, dividiéndolos en su parte real e

imaginaria. El siguiente procedimiento se lleva a cabo para cada una de las ventanas creadas anteriormente:

1) Creamos un objeto FFTSetup, que es un objeto que contiene información sobre la configuración de los cálculos de la FFT.
2) Convertirmos la ventana de float a DSPSplitComplex mediante la función `vDSP_ctoz` .
3) Establecemos la memoria necesaria para los cálculos requeridos para la FFT y comprobamos su disponibilidad, con la función `vDSP_create_fftsetup(log2n, FFT_RADIX2)` , que devuelve un objeto de tipo FFTSetup. Si no ha habido ningún error, podemos calcular la FFT.
4) Llevamos a cabo la FFT con la función `vDSP_fft_zrip` . Como parámetros de entrada, la función recibe la configuración FFTSetup, los datos de entrada, el número de puntos utilizado para la FFT, la dirección donde se guardará el resultado y el sentido de la transformada (directa o inversa).
5) Debido a la simetría de la FFT, la función entrega el resultado de forma empaquetada, de forma que los valores reales de la transformada, se agrupan en el primer elemento, como si de un elemento complejo se tratara, y sólo se conservan los valores complejos de índice menor a la mitad del número de puntos. Para proseguir con los cálculos, es necesario deshacer este empaquetamiento.
6) Finalmente, se procede al cálculo del módulo al cuadrado, para ello, se hace uso de la función `vDSP_zvcmul` , que calcula el producto conjugado de dos vectores. En nuestro caso, los dos vectores de entrada serán el mismo.

4.4.1.4 Bancos de filtros e integración

Antes de la fase de filtrado, se procede a la de normalización, buscando el máximo de la DEP y normalizando respecto a él. Para ello se hace uso de dos funciones: `vDSP_zvabs` ,que calcula el valor absoluto de un vector y `vDSP_maxmgv` , que devuelve el máximo de un vector.

Para calcular la salida del filtro, hacemos uso de la función `vDSP_mmul` , que realiza el producto de dos matrices: el filtro y la formada por los vectores hallados hasta el momento. Como aclaración, el filtro se incluye dentro de la carpeta de recursos en la aplicación, puesto que sus coeficientes corresponden a aquéllos calculados en Matlab, de este modo, si se cambia de tipo de filtro, sólo necesitaríamos cambiarlos en el fichero donde se guardan los actuales.

El resultado se guarda en una matriz de tipo *float*, para facilitar el siguiente paso.

4.4.1.5 Logaritmo

Para el cálculo del logaritmo, nos apoyamos en la función de C definida en *math.h*, *logf*, que calcula el logaritmo natural del número pasado como argumento.

Calculamos el logaritmo de todos los coeficientes hallados.

4.4.1.6 DCT

Finalmente, para el cálculo de la DCT, utilizamos el siguiente procedimiento:

1) Calculamos los coeficientes de la DCT, mediante las funciones de coseno y seno de C.
2) Calculamos la FFT siguiendo los pasos vistos en 4.4.1.3.
3) Multiplicamos los coeficientes de la DCT con las FFTs calculadas en 2) mediante la función `vDSP_zvmul` ,que multiplica dos vectores complejos.

Finalmente, del resultado de esta última operación obtenemos los coeficientes deseados. Estos se guardan en un array float de dimensiones *Nº de coeficientes* x *Nº de ventanas,* donde el número de coeficientes es 7 para nuestro caso particular. La función de extracción de MFCC se utiliza tanto en la interfaz de "Inicio" como en la de "Entrenamiento".

4.4.2 Integración

Para la implementación del bloque de integración, volvemos a hacer uso de funciones de la API vDSP. Como vimos en el capítulo 3, el método de integración que se decidió utilizar en la fase final de desarrollo es el de Media-Varianza. Recordemos que los resultados que entregaban eran satisfactorios y que, como aliciente, era el método más sencillo de implementar. Lo comprobaremos en este momento.

Tras haber calculado los coeficientes de Mel de una canción, llamamos al método de integración, con cabecera `(void)integracion: (int)metodo: (int)tamano:(int)salto: (int)longitud:(int)n_coef` , donde *metodo* se refiere al método utilizado para integrar, se dejó abierta la opción de implementar cualquier otro método; *tamano* es el tamaño de ventana de integración, *salto* es el salto de la ventana, *longitud* es el número de vectores de coeficientes de Mel y *n_coef* es el número de coeficientes utilizado.

El proceso de integración de un conjunto de vectores de coeficientes de Mel es el que sigue:

1) Creamos un array dinámico de dimensiones 2·Nº de coeficientes x Nº de ventanas. El valor 2 proviene del propio método de integración que, para cada ventana, crea un nuevo vector donde almacena la media y la varianza de los coeficientes.
2) Enventanamos según el número de muestras pasado por parámetros, en nuestro caso, 600 muestras.
3) Creamos un vector de longitud 2 x Número de coeficientes, mediante la función `vDSP_meanv` calculamos la media de los elementos de la ventana. Colocamos los resultados en la primera mitad del vector.
4) Haciendo uso de la función `vDSP_measqv` , que calcula la media de los valores al cuadrado, calculamos la varianza (Ecuación 4.1) y situamos los resultados en la segunda mitad del vector.

```
varianza = msv – powf(media, 2)
```
[4.1]

5) Si aún faltan datos por integrar volvemos a 2) , en caso contrario, hemos terminado la integración de los coeficientes de Mel.

Al contrario que pasaba en el desarrollo en Matlab, los procesos de etiquetado y de extracción de coeficientes-integración, son independientes. Como las características del sistema son conocidas a priori (número de coeficientes, tamaño de la ventana, de salto…) podemos ejecutar de forma independiente estos procesos. Ambos tienen en común el número de ventanas que resultan tras la integración, que es el mismo número de etiquetas que se utilizará. De este modo, se ejecutan de forma separada y posteriormente se unen sus resultados.

4.4.3 LIBSVM bajo iOS

Uno de los logros más importantes de este proyecto es la integración de la potente herramienta de clasificación basada en máquinas de soporte vectorial, LIBSVM, dentro de la aplicación iOS. De este modo, contamos con todas las posibilidades que la librería ofrece, siendo inabarcable el número de posibles aplicaciones futuras. En este apartado vamos a explicar cómo se integró la librería dentro del escenario de iOS y qué uso se hace de ella en esta aplicación en particular.

4.4.3.1 Integración

Como hemos mencionado en alguna ocasión durante este documento, el compilador de Xcode entiende cualquiera de estos tres lenguajes: C, C++ y Objective-C. Pues bien, aunque generalmente se utiliza su versión para Matlab, el núcleo de LIBSVM está escrito en C++.

Por lo tanto, para incluir LIBSVM en nuestra aplicación, basta con llevar a cabo estos dos pasos: incluir todas las clases del paquete dentro de la carpeta "Classes" (o "Other Sources") e incluir la siguiente línea en aquellas clases que vayan a hacer uso de la librería:

```
#import "svm.h"
```

que contiene las cabeceras de las clases contenidas en el paquete. Una vez seguidos estos pasos, el aspecto de Xcode debería ser algo similar a lo mostrado en la Figura 4.17.

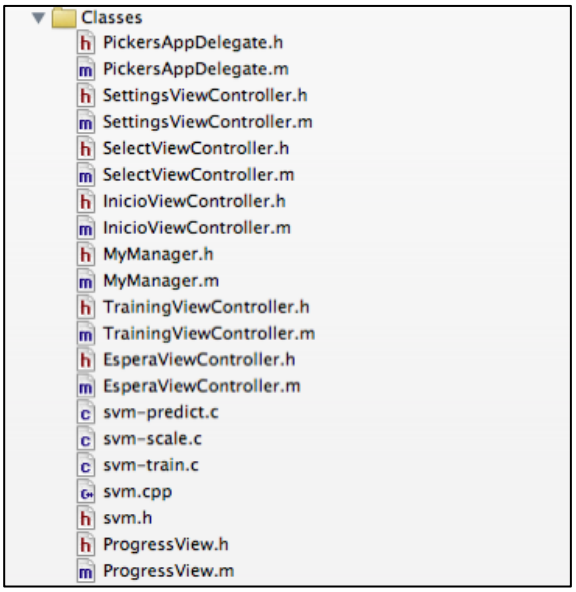

Figura 4.17. Integración de LIBSVM en PFC.

Tras la integración, es necesario explicar cómo hacer uso de las funciones de LIBSVM. El comportamiento de la librería es similar al que tiene cuando se le llama desde línea de comandos (la aplicación Terminal en Mac OSX) pero con una salvedad: debemos pasar los argumentos de entrada tal y como llegan a la función en concreto. De algún modo, nos estamos saltando el paso entre la línea de comandos y la función concreta, veamos un ejemplo que clarifique el proceso.

Llamada en Terminal a svm-train:

```
./svm-train -c 8 -g 0.25 train_repositorio
```

Misma llamada desde una clase de Xcode:

```
int j =  svmtrain(argc, argv);
```

Donde argc=6 es el número de argumentos que se le pasan a la función (incluyendo el nombre de la misma) y argv es un array de cadenas de texto con cada uno de los argumentos. Así.

argv = { “svmtrain”, “-c”, “8”, “-g”, “0.25”, “train_repositorio” }

De este modo, podemos hacer uso de cualquier función incluida dentro de LIBSVM. Además, para un desarrollo más completo, los mensajes de aviso de LIBSVM aparecerán en la “Debug Area”, que es la barra situada en la parte inferior de la ventana principal de Xcode.

4.4.3.2 Uso de LIBSVM en la aplicación PFC

Tras realizar una introducción al uso de las funciones de la librería LIBSVM, explicaremos dónde y cómo se utilizan éstas dentro de la aplicación PFC.

Como vimos en el apartado 4.3.1, la interfaz de Inicio tiene dos funciones principales: la reproducción de música y la clasificación de los fragmentos de música según las categorías establecidas. Para este último fin, hacemos uso de la funciones *svmtrain* y *svmpredict*, que se encuentran dentro de la librería LIBSVM

Estas funciones se apoyan en el uso de un conjunto de ficheros externos incluidos dentro de los documentos de la aplicación: *train_repositorio_def.dat*, *train_repositorio_custom.dat*, *modelo.model*, *test*, *train* y *output*. Los ficheros de extensión “.dat” guardan los vectores de coeficientes de Mel y su etiqueta ya sean correspondientes al modelo por defecto (con sufijo “_def”) o al personalizado (con “_custom”). Los ficheros sin extensión son propios de LIBSVM: *train* guarda los datos de entrenamiento en el formato apropiado, *test* contiene los datos para la clasificación y *output* está formado por la predicción del clasificador, mostrando una categoría por línea. Finalmente, *modelo.model* actúa de fichero intermedio entre las funciones de entrenamiento y predicción y guarda el modelo obtenido en la primera fase.

A continuación explicaremos el proceso seguido para el entrenamiento y la clasificación de los datos. Finalizada la fase de etiquetado, nos dirigimos a la pestaña de “Inicio”, donde se llevará a cabo la clasificación. En ese punto, contamos con un *array* compuesto por vectores de coeficientes de Mel integrados y etiquetas. Copiamos el contenido de dicho *array* en el fichero *train* en un formato entendible por la librería LIBSVM (*clase 1:valor 2:valor ... n:valor*, donde *n* es el número de características, en nuestro caso *n=14*). Posteriormente, llamamos a la función *svmtrain* de la siguiente forma:

```
int j =  svmtrain(9, argv);
```

donde

j: es un entero que especifica el resultado de la operación (ejecución sin errores o tipo de error, si procede).

9: es el número de argumentos que se le pasan a la función.

argv: es el array que contiene los argumentos y está formado por:

- `argv[0]="svmtrain"`: Es el nombre de la función
- `arg[1]…arg[6]`: Contiene las opciones seleccionadas en los ajustes: coste, gamma y *kernel*, ya sean las predefinidas o las personalizadas.
- `argv[7]=[filePath cStringUsingEncoding:NSUTF8StringEncoding]`: Ruta del fichero de entrenamiento, *train*.
- `argv[8]=[str_model cStringUsingEncoding:NSUTF8StringEncoding]`: Ruta del fichero donde se guardará el modelo de clasificación, *modelo.model*.

Tras esta llamada, dispondremos del modelo de clasificación, sin embargo, antes de proceder con ella, copiamos el vector de datos de test –de la canción que queremos clasificar –a un fichero que LIBSVM entienda, tal y como se hizo anteriormente. Una vez completado este paso procedemos a la clasificación a través de la llamada a *svmpredict*, llamada que tiene la siguiente forma:

```
int k =  svmpredict(4, argv2);
```

donde

k: es un entero que especifica el resultado de la operación (ejecución sin errores o tipo de error, si procede).

4: es el número de argumentos que se le pasan a la función.

argv2: es el array que contiene los argumentos. Está formado por:

- `argv2[0]="svmpredict"` : Es el nombre de la función.
- `argv2[1]=[filePathTest cStringUsingEncoding:NSUTF8StringEncoding]` : es el fichero *test* siendo filePathTest su ruta.
- `argv2[2]=[str_model cStringUsingEncoding:NSUTF8StringEncoding]` : Es el fichero con el modelo de clasificación obtenido en *`svmtrain`*; *`str_model`* se refiere a la ruta del archivo.
- `argv2[3]=[str_out cStringUsingEncoding:NSUTF8StringEncoding]` : Es el fichero de salida con las predicciones, *output*.

La función `cStringUsingEncoding` convierte una cadena de caracteres de Objective-C en una de C utilizando la codificación especificada, en este caso una representación de 8 bits.

Una vez obtenido el fichero de salida, se leen las predicciones y se almacenan en un vector para facilitar la reproducción y la muestra de las etiquetas correspondientes.

Como nota adicional, aclarar que el fichero *train_repositorio_def.dat* contiene alrededor de 2.000 vectores etiquetados y conforman la base del entrenamiento del modelo por defecto. Cada vez que arrancamos el programa, estos datos son cargados en memoria para añadirlos posteriormente a los nuevos datos de entrenamiento, si los hubiera. Adicionalmente, se implementó un mecanismo para que sólo se calculara el modelo cuando fuera estrictamente necesario, ahorrando así tiempo y recursos, de modo que, sólo se haga cuando alguna de estas condiciones ocurran: cambio de clasificador por defecto a personalizado (y viceversa), cambio de parámetros del clasificador o adición de nuevos datos de entrenamiento.

4.5 Casos de uso

En este apartado presentamos los casos de uso más comunes de la aplicación desarrollada.

En los diagramas, los iconos situados al lado de las flechas indican la interfaz a la que debe dirigirse el usuario para realizar las distintas acciones. Recordemos la relación icono-interfaz presentada al comienzo de este capítulo.

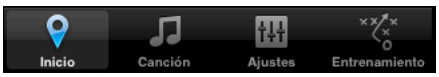

Figura 4.18. Leyenda para los diagramas de casos de uso

En primer lugar, la Figura 4.19 muestra el caso más sencillo de todos: la reproducción de música mientras se muestran las etiquetas. Este caso se da si el usuario no desea ni modificar el modelo de clasificación por defecto ni etiquetar más canciones para mejorarlo. En cambio, se presenta la posibilidad de reproducir otra canción distinta a la inicial, para ello deberá dirigirse a la interfaz "Canción".

En segundo lugar, en la Figura 4.21 presentamos un caso de uso más complejo, en el que incluimos etiquetado y entrenamiento, ya sea de un nuevo modelo personalizado o del modelo por defecto. En cualquiera de los casos, el diagrama concluye con la reproducción de música acompañada por la clasificación.

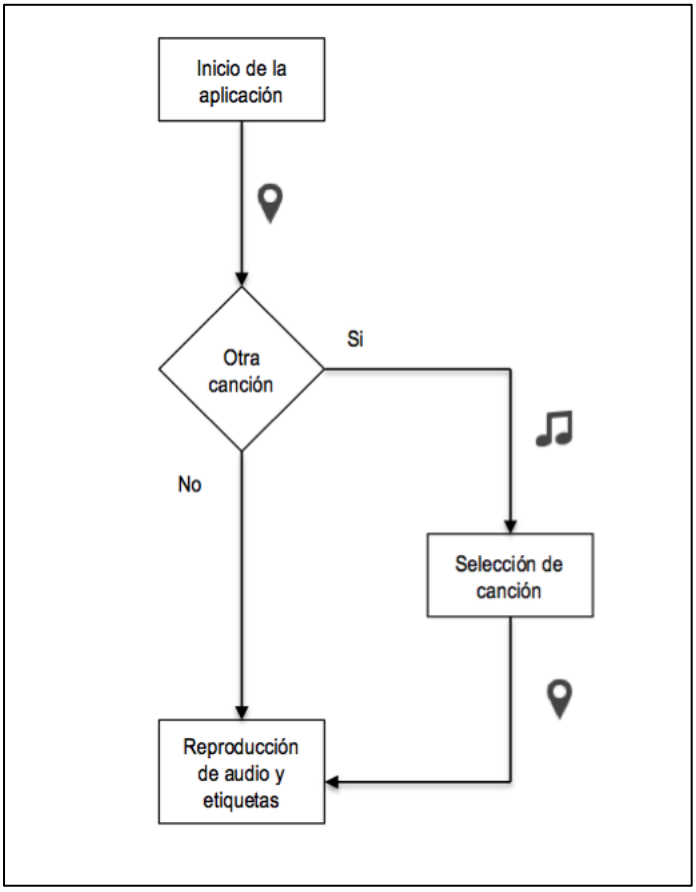

Figura 4.20. Primer caso de uso.

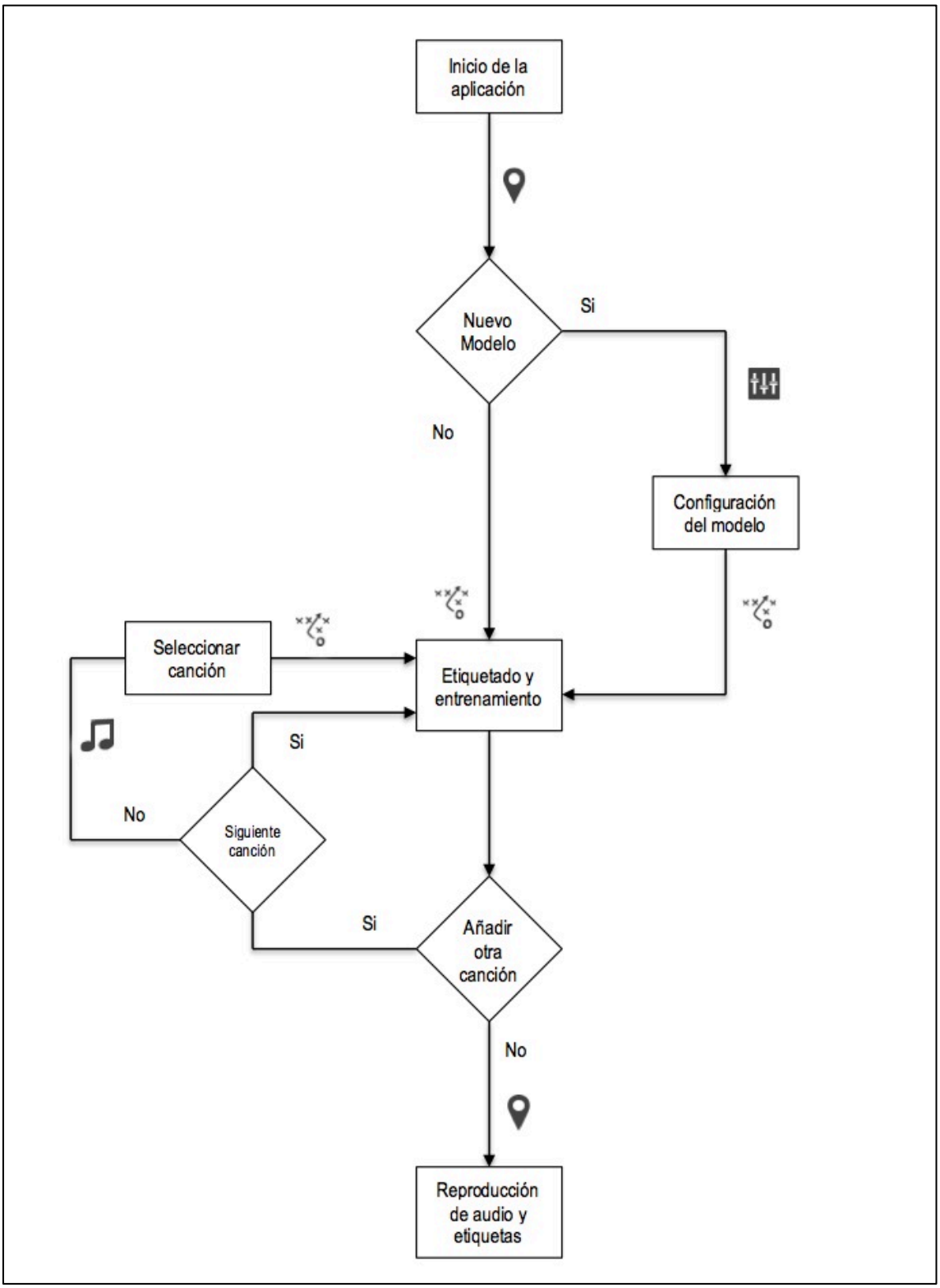

Figura 4.21. Segundo caso de uso

4.6 Selección de parámetros

Como aventuramos en la sección homónima 3.4, llegados a este punto los únicos parámetros que nos queda por ajustar en el sistema son los relativos al clasificador: la gamma y el coste. Se tomó como apropiado un kernel RBF, por lo que no formará parte del estudio.

Para llevar a cabo el proceso de búsqueda de los parámetros más convenientes, etiquetamos de nuevo las canciones pero esta vez en el propio dispositivo, bajo el programa PFC.app. Una vez obtenidos los resultados del etiquetado, se creyó conveniente volver a Matlab para la búsqueda, puesto que nos aportaba más facilidad en el manejo de tal cantidad de datos. El proceso seguido es el aconsejado en [10]. Emplearemos la búsqueda por "rejilla" dando valores del coste C y de la gamma, γ, usando validación cruzada. Se utilizan secuencias crecientes de valores exponenciales y se realizan dos búsquedas: en la primera se intenta identificar la mejor región –la región determinada por los C y γ que producen la mejor probabilidad de acierto –mientras que en la segunda se busca sobre esa región el par (C, γ) que entrega el mejor resultado.

En la primera búsqueda, se utilizaron los valores de C = 2^{-5}, 2^{-3}, …, 2^{15} y γ=2^{-15}, 2^{-13}, …, 2^{3}. Los resultados obtenidos se muestran en la Figura 4.22. En ella aparece delimitada por líneas discontinuas la región del plano formado por (C, γ) que proporciona mejores resultados (Nótese que los valores que aparecen en los ejes corresponden solamente a los exponentes a los que se eleva el 2). Vemos que en esa probabilidad de acierto ronda el 70%, con una nueva búsqueda, más exhaustiva sobre esa región intentaremos encontrar valores que mejoren esa cifra.

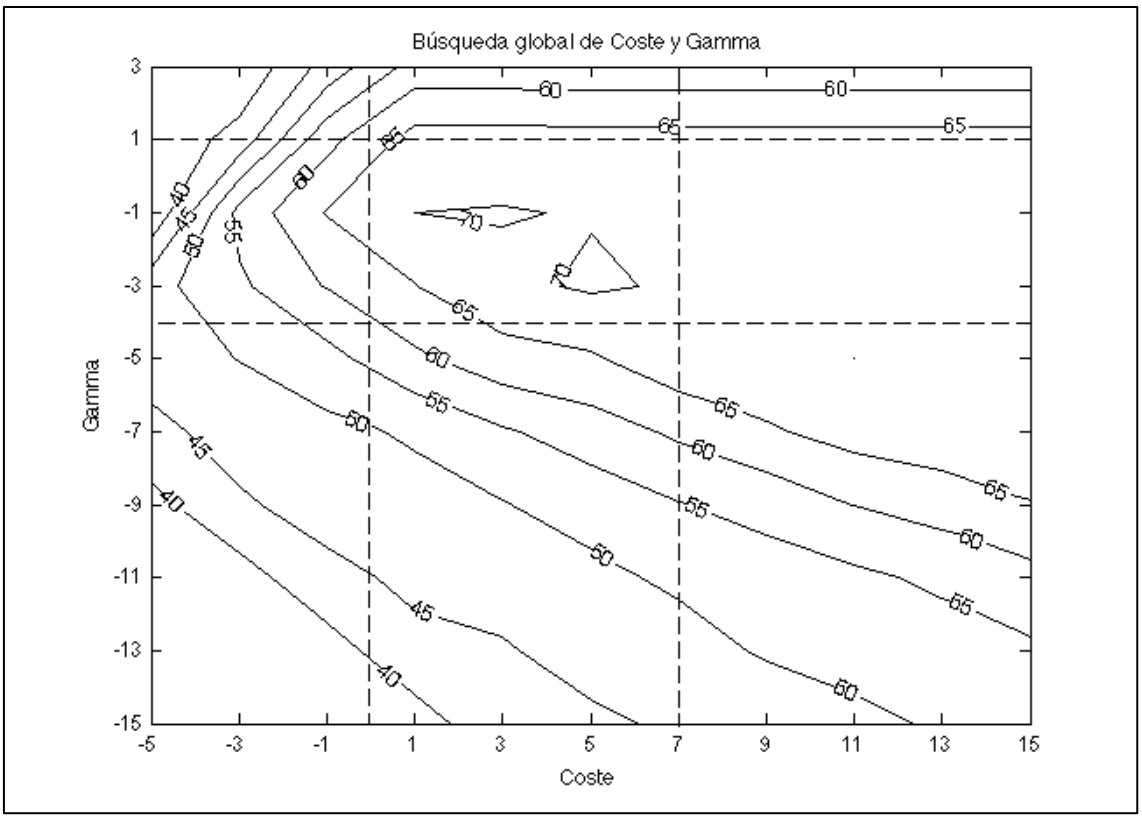

Figura 4.22. Búsqueda global de Coste y Gamma

Como se desprende de la Figura 4.22, la región para encontrar el mejor resultado es aquella delimitada por los valores de C=1 y C=2^7 y γ=2^{-4} y γ=2^5. Efectuamos pues la búsqueda en esa región, pero utilizando esta vez una menor separación entre valores, si en la primera búsqueda usábamos una separación entre exponentes de 2, ahora empleamos una de 0.25, consiguiendo una búsqueda más exhaustiva. Los resultados obtenidos se muestran en la Figura 4.23. Nos percatamos de que no existen valores de probabilidad de acierto mucho mayores de aquel 70% encontrado en la primera búsqueda, sin embargo, esta última búsqueda ha sido útil para encontrar el mejor resultado dentro de toda la rejilla. Este mejor resultado es una probabilidad de acierto del 71,4% y se consigue para C=2^3 y γ=2^{-2} . Estos valores serán incluidos en la aplicación y serán los valores por defecto que se cargarán. Consideramos que una probabilidad de acierto del 71,4% para validación cruzada es un resultado más que aceptable, si tenemos en cuenta que nuestro objetivo principal es la implementación en iOS de la aplicación de etiquetado y clasificación.

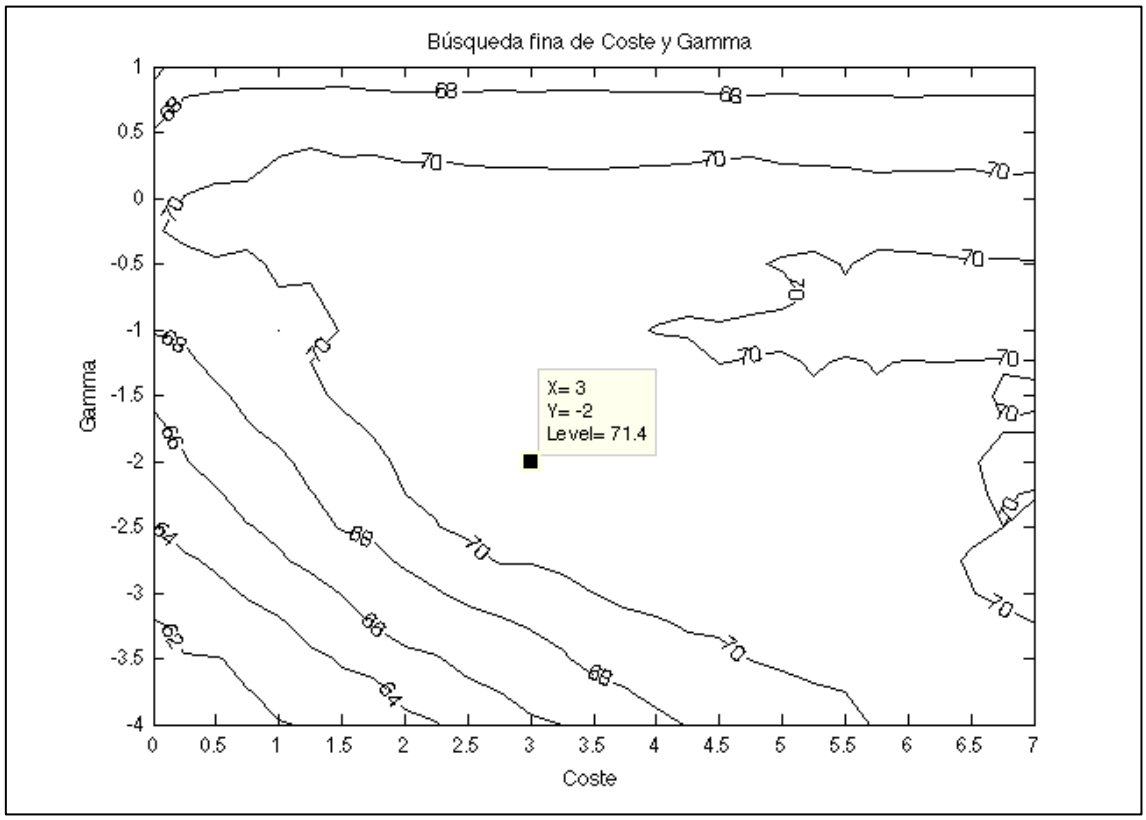

Figura 4.23. Búsqueda fina de Coste y Gamma

Tras la consecución del modelo, comprobamos su comportamiento con el conjunto de las canciones de test, obteniendo una probabilidad de acierto para dicho conjunto del 84,5%, una cifra muy satisfactoria. Por último, tratándose de un clasificador multiclase, resulta imprescindible comprobar cuál es el comportamiento del modelo recién obtenido al clasificar entre las tres distintas clases. Para ello, estudiaremos la matriz de confusión obtenida al clasificar los datos de test (Tabla 4.3). Recordemos que la diagonal recoge el número de aciertos, mientras que el resto de elementos de la columna muestra en detrimento de qué clase se ha equivocado el clasificador. El comportamiento del clasificador para la clase 1 ("Triste") es muy satisfactorio, un acierto del 95%, equivocándose en cuatro ocasiones. Para la clase 2 ("Normal"), la tasa de acierto es algo inferior, 90,2%, y comprobamos cómo el 78,5% de las ocasiones en que se equivoca es a favor de la clase 3. Por último, la clase con peor comportamiento es la 3, con una probabilidad de acierto del 70,4 %, equivocándose un 64,8% de las ocasiones a favor de la clase 1.

Número de predicciones		Clase real		
		1	**2**	**3**
Clase predicha	**1**	83	3	24
	2	1	129	13
	3	3	11	88

Tabla 4.3. Matriz de confusión de los datos de test.

A tenor de los resultados, consideramos que los valores de $C=2^3$ y $\gamma=2^{-2}$ nos ayudan a crear un modelo que satisface nuestras expectativas. A continuación, resumimos cómo quedaría el sistema de clasificación tras concluir la etapa de selección de parámetros iniciada en el capítulo anterior.

Bloque	**Parámetro**	**Valor**
Extracción de MFCC	Número de coeficientes	7
Integración	Tamaño de ventana	600 muestras
	Tamaño de salto	300 muestras
	Método de Integración	Media-Varianza
	Orden	N/A
Clasificador	Gamma	2^{-2}
	Coste	2^3
Probabilidad de Acierto	**Validación Cruzada**	**71,4 %**
	Datos de test	**84,5 %**

Tabla 4.4. Parámetros utilizados en el sistema final y resultados

5. Conclusiones

El proyecto que teníamos entre manos presentaba dos partes bien diferenciadas: la implementación de la aplicación OS que permitiera el etiquetado emocional de música y la búsqueda de un modelo de clasificación que lo permitiera.

De la primera de ellas –y más importante-, el objetivo era conseguir una aplicación que incluyera todos los mecanismos para etiquetar, entrenar, clasificar y reproducir música de tal manera que se mostrara por pantalla las emociones asociadas al fragmento de música escuchado en tiempo real. Creemos que no sólo cumplimos con el objetivo sino que, gracias a la inclusión de LIBSVM, conseguimos superarlo puesto que, finalmente se ha conseguido una aplicación con una interfaz atractiva que permite al usuario no sólo etiquetar, entrenar y clasificar bajo las tres categorías predefinidas sino que se le ofrece la posibilidad de incluir las suyas propias. Los logros a destacar de esta parte del trabajo son:

- Implementación de la extracción de los coeficientes de Mel en iOS.
- Integración de la librería para clasificación por máquinas de soporte vectorial LIBSVM dentro del entorno iOS.
- Creación de una interfaz para el etiquetado de fragmentos musicales.
- Creación de una interfaz para la reproducción de música y de etiquetas.

La segunda parte consistía en encontrar un modelo que clasificara la música según tres categorías emocionales: “Triste”, “Normal” y “Alegre”. El punto de partida era complicado puesto que presentaba dos dificultades. Por un lado, el carácter subjetivo de las categorías: es inmediato reconocer que, al tratarse de emociones, cada individuo puede percibir de distinta manera cada canción, ateniéndose a motivos que nada tienen que ver con lo técnico. Por otro lado, no resulta fácil etiquetar fragmentos de canciones ateniéndonos a las categorías citadas, puesto que no todas las piezas musicales inspiran tales emociones de forma tan inmediata. Sin embargo, estas dificultades se concibieron desde el primer momento como desafíos atractivos puesto que constituían elementos diferenciadores del proyecto.

Tras varias etapas de búsqueda de parámetros en distintos entornos (Matlab y Xcode), se consiguió configurar un modelo que entregaba unos resultados ampliamente satisfactorios. Primero establecimos el número de coeficientes de Mel que se iban a usar, 7, posteriormente establecimos 600 y 300 muestras como el tamaño de ventana y de salto, respectivamente, apropiados para la integración de los coeficientes de Mel, utilizando el método de la Media-Varianza. Finalmente, configuramos el clasificador con los valores de coste $C=2^3$ y $\gamma=2^{-2}$, finalizando así la puesta a punto de todo nuestro sistema.

Los resultados finales son satisfactorios, habiendo obtenido una probabilidad de acierto del 71,4% bajo validación cruzada y de un 84,5% al probarlo con los datos de test. Si bien estas probabilidades podrían ser mejoradas, no era objetivo prioritario de este proyecto centrarse en la consecución del modelo óptimo. En este sentido, la aplicación fue diseñada de tal manera que estuviera abierta para la futura implementación de nuevas funcionalidades (nuevos métodos de extracción de características, de integración,…), tal y como se verá en el siguiente capítulo.

6. Trabajo futuro

Como trabajo futuro propondremos varias mejoras de la aplicación así como nuevas funcionalidades que pueden ser interesantes. En primer lugar, parece un paso natural implementar todos los **métodos de integración** estudiados. Recordemos que los resultados obtenidos en Matlab nos hicieron concluir que el más apropiado era el de la Media-Varianza. Sin embargo, es posible que para otro tipo de aplicación cualquier modelo de los restantes sea más apropiado. Dentro del código de la aplicación se dejó la puerta abierta para una posible futura extensión.

Otra de las posibles modificaciones quedaría a expensas de la eliminación por parte de Apple de la restricción de usar las **canciones de la biblioteca musical** del usuario para ser tratadas. De este modo, la existencia de la interfaz "Canción" sería innecesaria, pues podríamos seleccionar directamente las canciones desde la interfaz propia del sistema. En caso de ocurrir tal situación, la implementación sería muy sencilla: bastaría con obtener las muestras PCM de la canción seleccionada y utilizarlas tal y como se utilizan en la versión actual de la aplicación.

Como vimos en el estudio de las distintas interfaces de la aplicación, las imágenes mostradas según la categoría del segmento reproducido son incluidas por defecto dentro de la carpeta de recursos. Parece interesante incluir una nueva función que permita al usuario elegir qué **fotografía** asocia a qué categoría. Se daría al usuario la posibilidad de elegir unas imágenes por defecto, las del carrete de fotos del dispositivo o incluso tomar las fotos en ese mismo momento. Existen multitud de ejemplos de aplicaciones que permiten el manejo de fotografías –muchos de ellos de la propia Apple –por lo que no sería complicado incluir esta nueva función.

Como cuarta mejora, se podría incluir una nueva funcionalidad que reprodujera todos los fragmentos de una misma categoría uno detrás de otro; por ejemplo, podríamos pedir a la aplicación que reprodujera todos los fragmentos clasificados como "alegres". Esta funcionalidad sería útil, por ejemplo, para la realización de vídeos o presentaciones multimedia, puesto que aligeraría el proceso de selección de música.

Finalmente, una nueva funcionalidad que podría ser muy interesante es la inclusión de capacidades para la **compartición de datos** etiquetados. El proceso de etiquetado puede ser

tedioso cuando el número de canciones es alto. Si dos usuarios, bajo las premisas de un mismo clasificador –mismas categorías –etiquetaran cada uno un grupo de canciones y se intercambiaran los datos generados, conseguirían crear un clasificador más robusto con menor esfuerzo. Si en lugar de dos, el número de usuarios fuera cuatro, los datos recopilados mejorarían aún más el clasificador. De modo que, conforme creciera el número de usuarios que compartiera sus datos de etiquetado las prestaciones aumentaran. Los dispositivos iOS están perfectamente capacitados para la comunicación a través de la red, por lo que sólo sería necesario encontrar las funciones adecuadas en la API y desarrollar un protocolo de comunicación entre ellos.

Como hemos visto, existe un amplio rango de posibles modificaciones y funcionalidades adicionales. Gracias a la integración de LIBSVM y a la plataforma de etiquetado, la aplicación diseñada en este trabajo puede servir como base a cualquier nueva aplicación.

A. Apéndice: Planificación y presupuesto

A.1 Planificación del proyecto

A continuación se muestra un diagrama de Gantt donde se aprecia la planificación temporal del proyecto (Figura A.1)

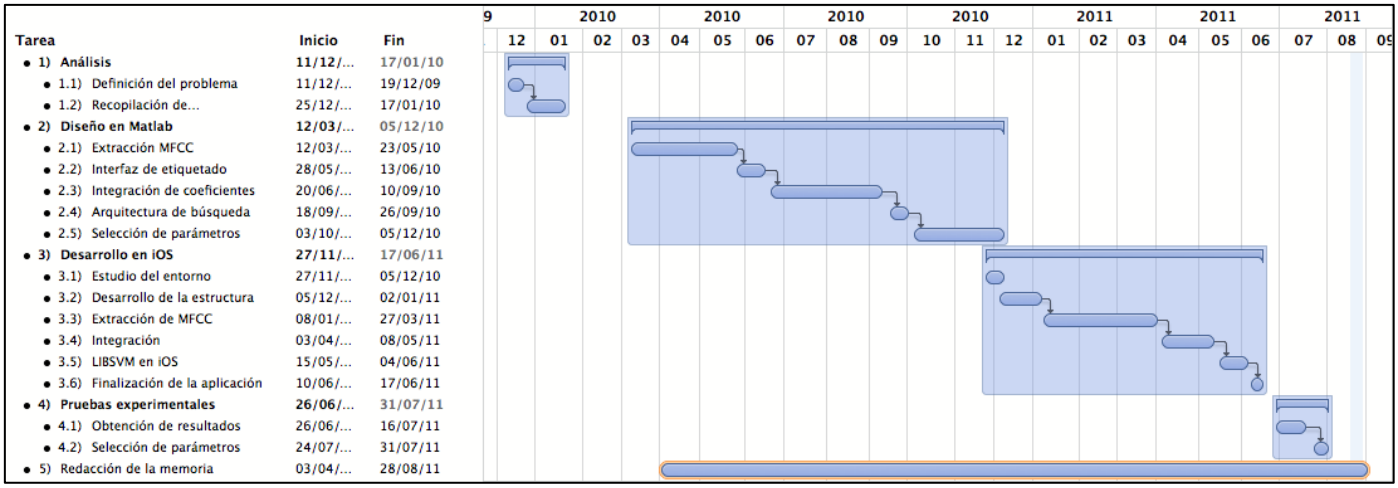

Figura A1. Diagrama de Gantt de la planificación

A.2 Presupuesto del proyecto

Este apéndice está dedicado a la elaboración del presupuesto asociado a la realización del presente proyecto. Para hacer la estimación de este importe económico se tiene en cuenta tanto el coste de los medios materiales como el coste de los recursos humanos empleados para su realización.

A.2.1 Coste de los medios materiales

En la Tabla A.1 se hace un desglose del coste unitario del material utilizado para realizar el proyecto. Al total del coste de material utilizado hay que añadir el coste del emplazamiento de trabajo donde se ha llevado a cabo la realización del proyecto. El lugar de trabajo está debidamente acondicionado mediante alumbrado, calefacción, aire acondicionado, servicio de limpieza, mesas, sillas, tomas de corriente, y red de datos de banda ancha. Su coste en alquiler se estima en 750 euros por mes. Se ha estimado una duración del proyecto en 21 meses, por lo que el coste en concepto de alquiler de emplazamiento puede estimarse en 15.750 euros.

CONCEPTO	COSTE UNITARIO (EUROS)
Ordenador personal. iMac de Apple de 27'' y 2,7 GHz	1.649
Licencia Matlab	2.400
Licencia de desarrollador de iOS	68
iPod Touch de 8 GB	229
Impresora + Tóner + Papel	170
Gastos de oficina	130
Encuadernación de los tomos	180
TOTAL MATERIAL	**4.826**

Tabla A.1. Coste unitario del material utilizado en la realización del proyecto.

Sumando el coste total del material más el alquiler del lugar de trabajo se obtiene que: el coste total de medios es de 20.576 euros.

A.2.2 Coste del personal

Según el Colegio Oficial de Ingenieros de Telecomunicación (COIT), el sueldo medio de un Ingeniero de Telecomunicación es de 30.000 euros brutos anuales. Sin embargo, en este caso como referencia podemos escoger un sueldo inicial para un trabajador en fase de realización del proyecto de fin de carrera o recién titulado, que se establece en 18.000 euros brutos anuales (incluido beneficios sociales). En dichas condiciones, el salario ronda los 1.500 euros brutos al mes, valor que se tomará como referencia. Por lo tanto, el coste del proyecto por coste de personal asciende a un total de 31.500 euros.

A.2.3 Coste de la dirección

Para la realización este proyecto de ingeniería, ha participado una persona junto con su tutor, que ha realizado una labor de apoyo y consultoría.

El salario del tutor se estima, de forma general, como un 7 % de la suma del coste de material más el coste de la mano de obra hasta un límite de 30.000 euros anuales. Por tanto el coste de la dirección del proyecto a lo largo de la elaboración de este estudio es de 3.645,32 euros.

A.2.4 Coste total del proyecto

El importe económico final del proyecto se calcula como la suma del coste de los medios materiales más el coste del personal y dirección del proyecto. Realizando este cálculo se obtiene que el coste total del proyecto asciende a 55.721,32 euros.

Leganés a 1 de Septiembre de 2011

El ingeniero proyectista

Fdo. Antonio José Blanco Calle

B. Apéndice: Lista de canciones

A continuación se muestran las canciones que se han utilizado como datos de entrenamiento y de test. Todas estas canciones forman parte de la biblioteca musical del autor y fueron obtenidas bien en formato digital a través de la iTunes Store de Apple o como copia de seguridad de un soporte físico.

Nombre	Artista	Álbum
Undercover of Darkness	The Strokes	Angles
Dog Days Are Over	Florence + The Machine	Between Two Lungs
Married Life	Michael Giacchino	Up
Cornerstone	Arctic Monkeys	Humbug
Good Vibrations	The Beach Boys	Sounds of Summer
I Walk The Line	Johny Cash	Super Hits
Bohemian Rhapsody	Queen	A Night At The Opera
Waterloo	ABBA	Gold
Hole In My Soul	Aerosmith	Nine Lives
Ironic	Alanis Morissette	Jagged Little Pill
Adagio	Tomaso Albinoni	Apollo Classics-Baroque
No Matter What	Boyzone	By Request
Sad Eyes	Bruce Springsteen	Greatest Hits
Run To You	Bryan Adams	The Best Of Me
Suspicious Minds	Elvis Presley	Elvis Presley Star Profile
Walking on Sunshine	Katrina And The Waves	Walking On Sunshine
Everybody's Changing	Keane	Hopes & Fears
Leaving So Soon?	Keane	Under The Iron Sea
Suddenly I See	KT Tunstall	Eye To The Telescope
Parklife	Blur	Parklife
No Distance Left To Run	Blur	13
I Guess That's Why They Call It The Blues	Elton John	Greatest Hits 1970-2002
Alright	Supergrass	In It For The Money
Shine	Take That	Beautiful World
Back For Good	Take That	Greatest Hits

My Girl	Temptations	Reflections
Setting Sun	The Chemical Brothers	Singles 93-03
I can't be with you	The Cranberries	Stars
Walk Idiot Walk	The Hives	Tyrannosaurus Hives
Sunday Bloody Sunday	U2	WAR
When The Sun Goes Down	Arctic Monkeys	Whatever People Say I Am, That's What I'm Not
Mardy Bum	Arctic Monkeys	Whatever People Say I Am, That's What I'm Not
A Certain Romance	Arctic Monkeys	Whatever People Say I Am, That's What I'm Not
Sweet Child O' Mine	Guns N' Roses	Greatest Hits
Since I Don't Have You	Guns N' Roses	Greatest Hits
Celebrity Skin	Hole	Celebrity Skin
Nessun Dorma Turandot (Puccini)	Luciano Pavarotti	Pavarotti Forever
'o Sole Mio	Luciano Pavarotti	Pavarotti Forever
God Knows	Mando Diao	Hurricane Bar
The Prettiest Thing	Norah Jones	Feels Like Home
Entre Dos Aguas	Paco De Lucía	Antología
Time For Heroes	The Libertines	Up The Bracket
How Soon Is Now	The Smiths	The Very Best Of The Smiths
I Am The Resurrection	The Stone Roses	The Very Best Of
The Modern Age	The Strokes	Is This It
Sonnet	The Verve	Urban Hymns
Hotel Yorba	The White Stripes	White Blood Cells
Videotape	Radiohead	In Rainbows
House Of Cards	Radiohead	In Rainbows
Time Of The Season	The Zombies	Esta Noche Cruzamos El Mississippi
Si Tú No Vuelves	Miguel Bosé & Shakira	Papito
Under The Bridge	Red Hot Chili Peppers	Greatest Hits
Scar Tissue	Red Hot Chili Peppers	Greatest Hits
Otherside	Red Hot Chili Peppers	Greatest Hits
Universally Speaking	Red Hot Chili Peppers	Greatest Hits
Lagrimas Negras	Bebo & Cigala	Lagrimas Negras
Starlight	Muse	Black Holes & Revelations
Barco A Venus	Mecano	Grandes Éxitos
Me Colé En Una Fiesta	Mecano	Grandes Éxitos
Maquillaje	Mecano	Grandes Éxitos
Karma Chameleon	Culture Club	Greatest Hits
Friends to Go	Paul McCartney	Chaos And Creation In The Backyard
Rearviewmirror	Pearl Jam	Rearviewmirror
Last Kiss	Pearl Jam	Rearviewmirror
Yellow Ledbetter	Pearl Jam	Rearviewmirror
Personal Jesus	Johnny Cash	American IV: The Man Comes Around

Bachelorette	Bjork	Vespertine
Mediterráneo	Juan Manuel Serrat	Nuestra Mejor Cancion
This Modern Love	Bloc Party	Silent Alarm
I Just Called To Say I Love You	Stevie Wonder	Definitive Collection
Fields Of Gold	Sting & The Police	The Very Best Of Sting & The Police
Beautiful Ones	Suede	Coming Up
Águas de Março	Elis Regina & Tom Jobim	Elis & Tom
Men's Needs	Kate Nash	NME Awards Compilation
I Saved The World Today	Eurythmics	The Chosen 1
I Shot The Sheriff	Bob Marley & The Wailers	Legend
Where'er Ye Go	Paul Weller	22 Dreams
Shiver	Coldplay	Parachutes
I feel you	Depeche Mode	The Singles 81>85
Contigo	Joaquín Sabina	Yo, mi, me, contigo
Time To Pretend	MGMT	Oracular Spectacular
My Baby Just Cares For Me	Nina Simone	The Very Best Of Nina Simone

Tabla B.1. Lista de canciones de entrenamiento y test.

Bibliografía

[1] Saráchaga, G., Sartori, V., & Vignoli, V. *Identificación Automática de Resúmenes de Canciones.* Proyecto Fin de Carrera, Universidad de la República, Facultad de Ingeniería, Montevideo.

[2] Nóbrega, R., & Cavaco, S. *Detecting key features in popular music: case study - singing voice detection.* Universidade Nova de Lisboa, CITI, Faculdade de Ciências e Tecnologia, Caparica.

[3] Foote, J. (1997). Content-based retrieval of music and audio. *Multimedia Storage and Archiving Systems II , 3229*, 138-147.

[4] Sigurdsson, S., Brandt Petersen, K., & Lehn-Schioler, T. *Mel Frequency Cepstral Coefficients: An Evaluation of Robustness of MP3 Encoded Music.* Technical University of Denmark, Informatics and Mathematical Modelling, Lyngby.

[5] Wankhammer, A., Sciri, P., & Sontacchi, A. (2009). *Chroma and MFCC based pattern recognition in Audio Files utilizing Hiden Markov Models and Dynamic Programming.* Univ. of Music and Performing Arts, Inst. of Electronic Music amd Acoustics, Graz.

[6] Meng, A., Ahrendt, P., & Larsen, J. (2005). Improving music genre classification using short-time feature integration. *Proc. ICASP* , 497-500.

[7] Tzanetakis, G., & Cook, P. (2002). Musical genre classification of audio signals. *IEEE Trans. Speech Audio Process , 10* (5), 293-302.

[8] Meng, A., Ahrendt, P., & Larsen, J. (2007). Temporal Feature Integration for Music Genre Classification. *IEEE Trans. on Audio, Speech, and Language Processing , 15* (5), 1654-1663.

[9] Alba, J. *Decisión, estimación y clasificación.* Universidad de Vigo, Departamento de Teoría de la Señal y Comunicaciones.

[10] Hsu, C.-W., Chang, C.-C., & Lin, C.-J. (2000). *A Practical Guide to Support Vector Classification.* National Taiwan University, Department of Computer Science.

[11] Chang, C.-C., & Lin, C.-J. (2010). *LIBSVM: a Library for Support Vector Machines.* National Taiwan University, Department of Computer Science.

[12] Angulo, C. *Aprendizaje con máquinas núcleo en entornos de multiclasificación.* Universitat Politècnica de Catalunya, Departament d'Enginyeria de Sistemes, Automàtica i Informàtica Industrial.

[13] *Smartphone*. (2011). Obtenido de Wikipedia: http://en.wikipedia.org/wiki/Smartphone

[14] *Wikipedia*. (2011). Obtenido de Mobile Application Development: http://en.wikipedia.org/wiki/Mobile_application_development

[15] *Edible Apple*. (2010). Obtenido de Number of iOS developers vs Android Developers: http://www.edibleapple.com/number-of-ios-developers-vs-android-developers/

[16] *Wikipedia*. (2011). Obtenido de Mobile Operating System: http://en.wikipedia.org/wiki/Mobile_operating_system

[17] *Mobclix*. (2011). Obtenido de Mobclix Index: Monthly Value of an App User: http://blog.mobclix.com/2011/02/10/mobclix-index-monthly-value-of-an-app-user/

[18] Apple. (2010). *iOS Developer Library*. Obtenido de iOS Technology Overview: http://developer.apple.com/library/ios/#documentation/Miscellaneous/Conceptual/iPhoneOSTechOverview/Introduction/Introduction.html

[19] Apple. (2010). *Mac OS X Developer Library*. Obtenido de The Objective-C Programming Language: http://developer.apple.com/library/mac/#documentation/Cocoa/Conceptual/ObjectiveC/Introduction/introObjectiveC.html

[20] *Wikipedia*. (2011). Obtenido de Objective-C: http://es.wikipedia.org/wiki/Objective-C

[21] Mark, D., & LaMarche, J. (2009). *Beginning iPhone 3 Development.* Apress.

[22] *Stack Overflow*. (s.f.). Obtenido de http://stackoverflow.com/